High-level Programming Languages and Systems for Cyber-Physical Systems

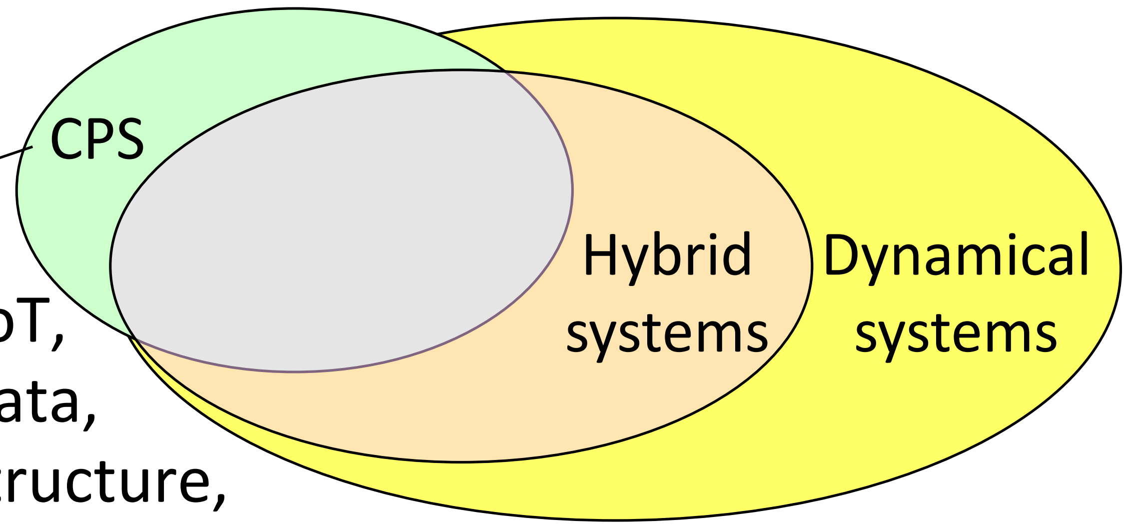
Kazunori Ueda

Waseda University, Tokyo, Japan

Halmstad Summer School of Cyber-Physical Systems, July 17-21, 2017

Background

- Cyber-physical systems (CPS, 2000's–) = systems with *computational* and *physical* components
- Hybrid systems (1990's–) = dynamical systems with *continuous* and *discrete* behavior



Various aspects:

- embedded systems, IoT, sensor network, big data, social/network infrastructure, distributed computing, security, …

Computational foundations for

- interacting with the physical world (= implementing CPSs)
- modeling, simulation and verification

Computing/modeling paradigms for CPSs

- Key issue
  = modeling of, and interfacing with, the *physical* world

Physical systems

$$\frac{d^2x}{dt^2} = 10$$

Computer systems

$$x_{t+1} = 1 - x_t$$
$$y_{t+1} = 2\, y_t$$

- Continuous (+ discrete) domain
- Math with differential (+ algebraic) equations
- Time

↔

- Discrete domain
- Programming languages
- Algorithms
- Abstraction

How to reconcile them with computing abstraction of physical systems?

Computing/modeling paradigms for CPSs

- Edward A. Lee: "Cyber-Physical Systems: Are Computing Foundations Adequate?"

  NSF Workshop On Cyber-Physical Systems, October, 2006

4. Research directions
   - Putting time into programming languages
   - Rethinking the OS/programming split
   - Rethink the hardware/software split
   - Memory hierarchy with predictability
   - Memory management with predictability
   - Predictable, controllable deep pipelines
   - Predictable, controllable, understandable concurrency
   - Concurrent components
   - Networks with timing
   - Computational dynamical systems theory

Hybrid systems

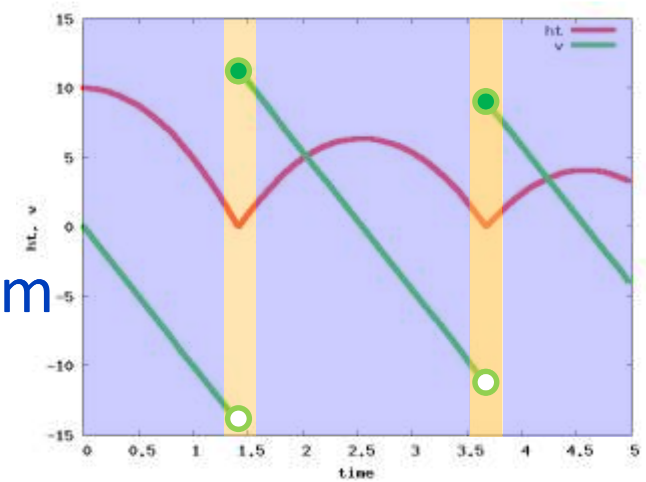
- Systems whose states can make both continuous and discrete changes

  Examples:
  - bouncing ball, billiard, . . .
  - thermostat + air conditioner + room
  - traffic signals + roads + cars

  In general:

  Dynamical systems whose description involves case analysis

  - physical, biological, control, cyber-physical, etc.



- Relates to computer science, control engineering and apps.
- Programming language aspects rather unexplored

Challenges and questions

- Designing and implementing programming/modeling languages for hybrid systems
  - What are the basic notions and constructs?
    cf. automata (concrete) vs. λ-calculus (abstract)
  - Are they simple and accessible to non-specialists (e.g., engineers outside CS) ?
- Language constructs are divided into
  - those determining the underlying computational model (primitives)
  - those motivated by software engineering point of view (user language)

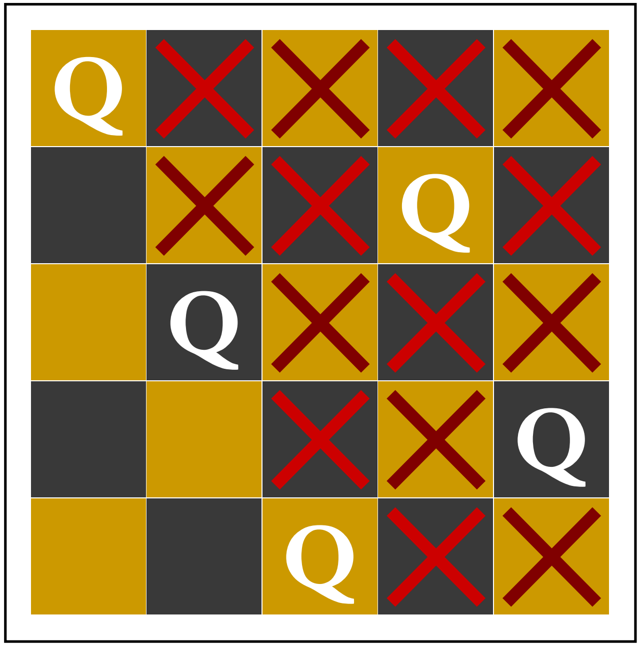
Modeling frameworks for hybrid systems

- Hybrid Automata and other “hybrid” models (Petri nets, I/O automata, Process Algebra, etc.)
- Modeling languages and tools with equations and updates
  - Modelica, Acumen, Ptolemy, Hybrid Language, ...
- *Constraint-based* languages and tools (domain = functions over time)
  - iSAT (Boolean+arithmetic constraint solver)
  - Hybrid CC (hybrid concurrent constraint language)
  - CLP(F) (constraint LP over real-valued functions)
  - Kaleidoscope ’90 (discrete time)
  - HydLa (constraint hierarchy)

L. P. Carloni et al, Languages and Tools for Hybrid Systems Design, *Foundations and Trends in Electronic Design Automation*, Vol.1 (2006), pp.1-193.

Constraint Programming (CP)

- A declarative programming paradigm in which a problem is described using equations/inequations over continuous or discrete domains



x_1 x_2 x_3 x_4 x_5

- Variables: $x_1, \ldots, x_5$
- Domain: $1 \le x_i \le 5$
- Constraints:
  if $i \neq j$ then
  - $x_j \neq x_i$
  - $x_j \neq x_i + |j - i|$
  - $x_j \neq x_i - |j - i|$

Constraint Programming (CP)

- Features and essence
  - *No algorithms*: CP languages are often called modeling languages
  - Developed in AI and Logic Programming communities
    - where the central interest has been constraint satisfaction and constraint propagation
    - many libraries for mainstream languages
    - CP languages are mostly based on Logic Programming
  - Another view of CP: *computing with partial information*
    - by means of symbolic execution

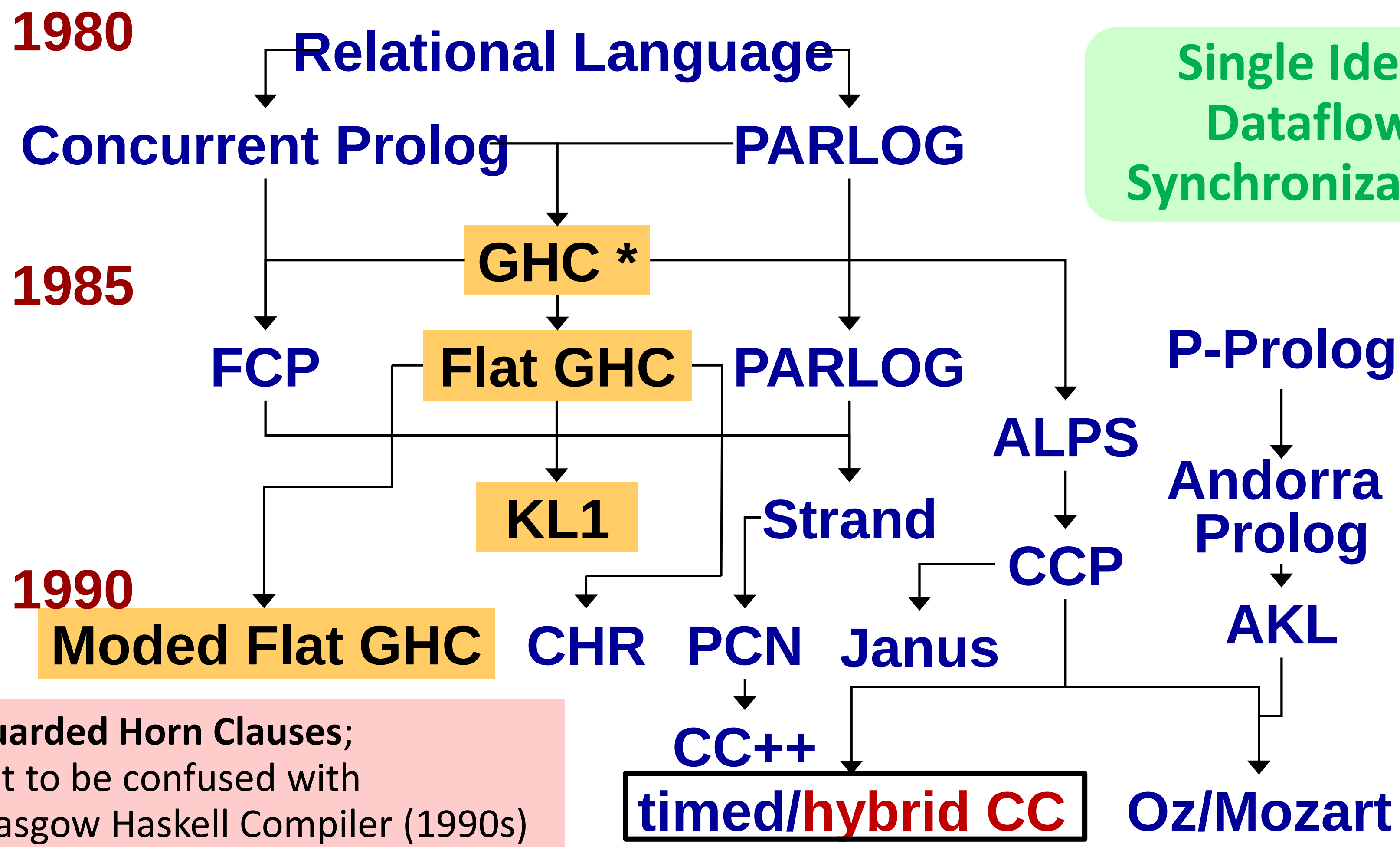
Constraint Programming (CP)

- Different flavors and applications
  - Constraint satisfaction problems (CSPs)
    - Domains: finite, real, interval, ...
  - SMT (satisfiability modulo theories)
    - complex combination of logical connectives
    - usually not compute most general solutions
  - (Constraint-based) Concurrency
    (a.k.a. Concurrent Constraint Programming)

Communication:	**tell**ing and **ask**ing of constraints	
Synchronization:	$\Rightarrow$	(also for conditionals)
Composition:	$\wedge$	
Hiding:	$\exists$	(also for fresh name creation)

Early history of constraint-based concurrency

Originated by process interpretation of logic programs



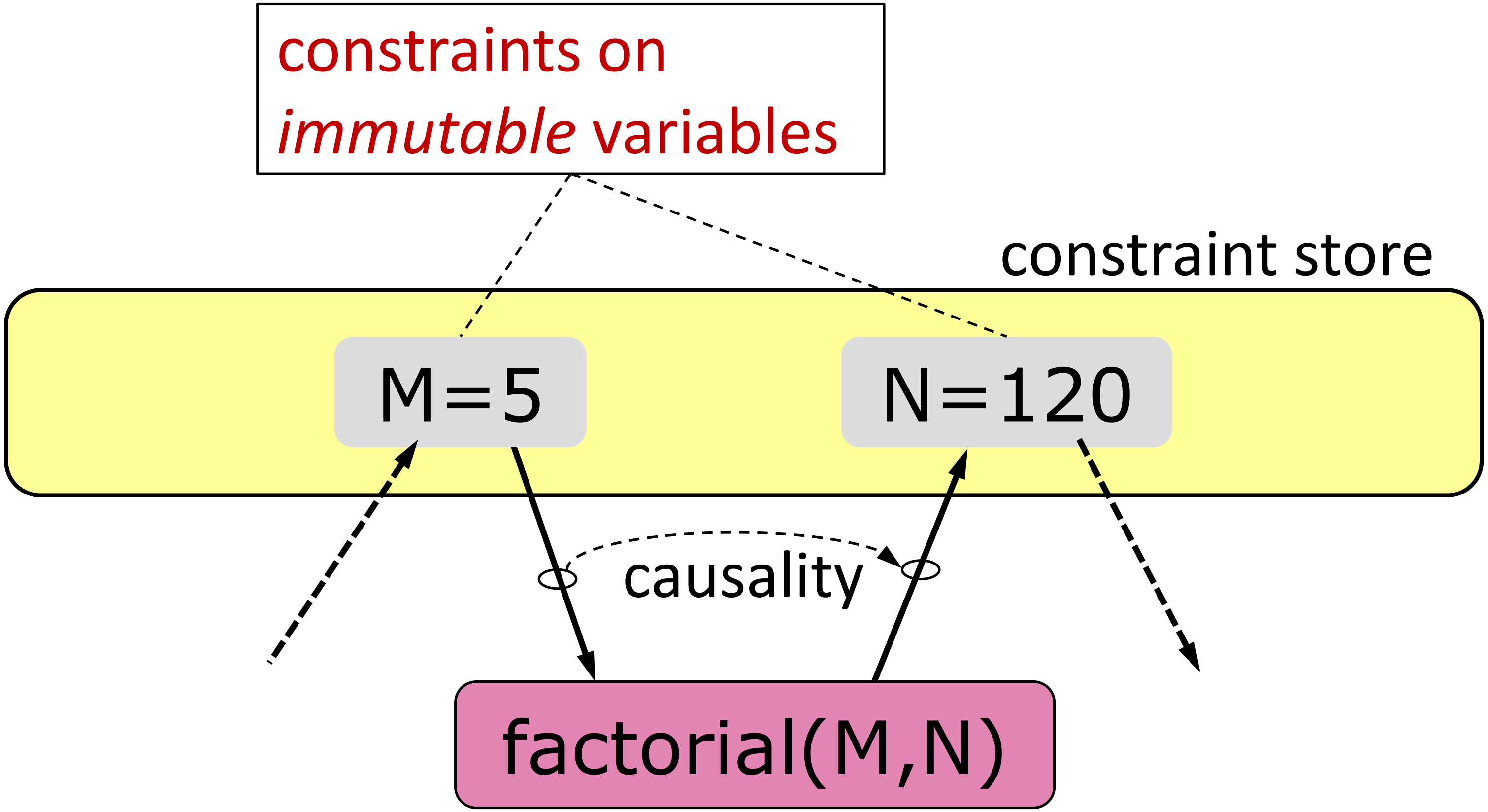
Single Idea: Dataflow Synchronization

* **Guarded Horn Clauses**; not to be confused with Glasgow Haskell Compiler (1990s)

Kazunori Ueda: Logic/Constraint Programming and Concurrency: The Hard-Won Lessons of the Fifth Generation Computer Project. *Science of Computer Programming*, 2017

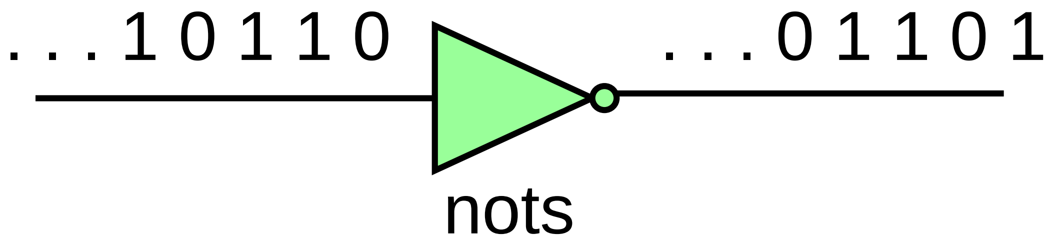
Constraint-based concurrency

```
factorial(X,Y) :- X=:=0 | Y:=1.
factorial(X,Y) :- X > 0 |
  X1:=X−1, factorial(X1,Y1), Y:=X*Y1.
```



Constraint-based concurrency

- Inverter accepting a sequence of input data




```
nots([],     Y  ) :- true | Y=[].
nots([0|X],Y0) :- true | Y0=[1|Y], nots(X,Y).
nots([1|X],Y0) :- true | Y0=[0|Y], nots(X,Y).
```

- Discrete event systems can be represented using possibly infinite lists.
  - e.g., [0,1,1,0,1|A]

Constraint-based concurrency

- Constraints imposed by “nots(X,Y)”:

Observed	Published	Rest
X=[0,1,1,0,1]	Y=[1,0,0,1,0]	(none)
X=[]	Y=[]	(none)
X=[0,1,1,0,1|X’]	Y=[1,0,0,1,0|Y’]	nots(X’,Y’)
(none)	(suspending)	nots(X,Y)
X=[2|_]	(reduction failure)	
X=[0|_], Y=[0|_]	(Inconsistency)	

Constraint Programming for hybrid systems

- Declarative description of hybrid systems
  = constraint programming of functions over time
  - cf. constraint programming over infinite sequences
- Many features are inherited from constraint-based concurrency
  - Implication (⇒) for synchronization and conditionals
  - Conjunction (∧) for parallel composition
  - Existential quantification (∃) for hiding

$$\Box(\underbrace{\text{e-stop} = 1}_{\text{(ask)}} \;\Rightarrow\; \underbrace{\text{speed}' = -4.0}_{\text{(tell)}})$$

Challenges from the language perspective

- Establish a high-level programming/modeling language
  - equipped with the notion of *continuous time*,
  - equipped with the notion of *continuous changes*,
  - that properly handles *uncertainties* and *errors of real values*,
  - that properly handles *conditional branch* under uncertainties and errors of real values,
  - equipped with constructs for *abstraction* and *parallel composition*.
  - etc.
- Establish semantical foundations
- Establish implementation technologies

Rigorous simulation

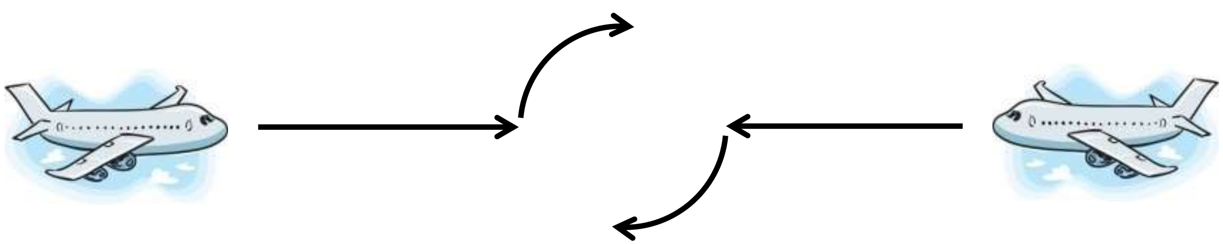
- Computers were born for numerical simulation, and simulation (in a broad sense) is still an important application of high-performance computers for the design and analysis of all kinds of systems.
- "How (much) can we trust these simulation results?"
    - For some simple problems, ordinary simulation with a standard tool *cannot* yield a single significant digit.

Rigorous simulation

- Simulation of hybrid systems is particularly hard and can easily go qualitatively wrong (due to conditional branch). A technique for rigorous simulation is very important.

Collision of three bodies

- Some CPSs are *safety-critical* or *mission-critical* also.



Collision avoidance model

Rigorous simulation vs. verification

- Most research on hybrid systems aims at verification as *decision problems*
  - yes/no answer (i.e., whether it works)
  - possibly with counterexamples (i.e., why it doesn't work)
- Rigorous simulation will require less from you and tell you more
  - no proof skills (cf. interactive theorem solving)
  - no proof goals (cf. automatic verifier)
    - still can be used to prove something (e.g., W. Tucker's proof on Lorenz attractors, R. E. Moore Prize 2002)
  - (often visualized) trajectories (i.e., how it works)
  - error margin (i.e., how safe it is)

HydLa : Overview and features (1/4)

- The field of hybrid systems comes with many notations, concepts and techniques; rather difficult to get into.
- Our challenge is to see whether a rather simplistic formalism can address various aspects of hybrid systems
- Goals:
  - Identifying computational mechanisms
  - Modeling and *understanding* systems that are not large but may exhibit problematic behavior
- Non-goals (currently):
  - Modeling large-scale systems

HydLa : Overview and features (2/4)

- Declarative (↔ Procedural)
  - Minimizes new concepts and notations by employing popular mathematical and logical notations
    - $=,\ \leq,\ +,\ \times,\ \frac{d}{dx},\ \wedge,\ \Rightarrow,\ \Leftrightarrow$, ...
  - Describes systems as logical formulae with hierarchy
    - No algorithmic constructs such as states and state changes, iteration, transfer of control, etc.
  - Still, it turns out that the semantics comes with large design space, e.g.,
    - how to compare two uncertain values?
    - what continuity should we assume?

HydLa : Overview and features (3/4)

- Constraint-based
  - Basic idea: defines functions over time using constraints including ODEs, and solves initial value problems
    - cf. streams are defined by difference equations
  - Handles partial (incomplete) information properly
    - Intervals (e.g., $x \in [1.0, 3.5]$) fit well within the constraint-based framework
    - Allows modeling and simulation of *parametric* hybrid systems
  - Symbolic computation based on *consistency* checking
    - Powered by numerical techniques

HydLa : Overview and features (4/4)

- Features constraint hierarchies (Alan Borning, 1992)
  - Motivation: It's often difficult to describe systems so that the constraints are consistent and well-defined.

    Examples: bouncing ball (, billiard, . . .)
    - A ball normally obeys the law of gravity (default), while it obeys the collision equation when it bounces (exception).
    - The frame problem (McCarthy and Hayes, 1960s) occurs in the description of complex systems.
      - We can't enumerate all possible exceptions
  - Want to define these properties concisely and in a modular manner.

Example 1 : Sawtooth function

$\text{INIT} \Leftrightarrow f = 0.$

$\text{INCREASE} \Leftrightarrow \square(f' = 1).$

$\text{DROP} \Leftrightarrow \square(f- = 1 \Rightarrow f = 0).$

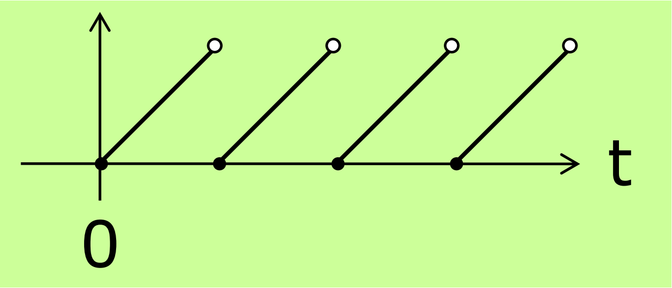
INIT, (INCREASE << DROP).

rules

guard

priority

- Describes properties at time 0.
- Time argument is implicit: $\square(f' = 1)$ means $\forall t \geq 0\ (f'(t) = 1)$
- $f-$ stands for the left-hand limit of f.



Example 1b : Sawtooth function

INIT ⇔ $0 \le f < 1$.

INCREASE ⇔ $\square(f' = 1)$.

DROP ⇔ $\square(f^{-} = 1 \Rightarrow f = 0)$.

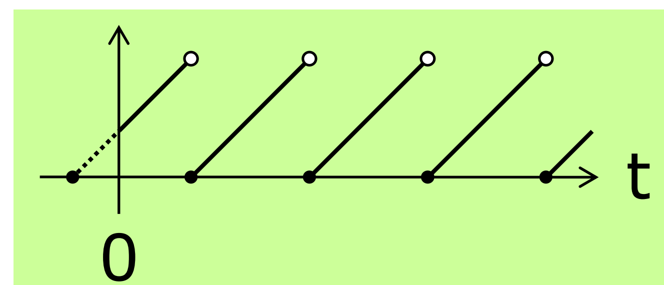
INIT, (INCREASE << DROP).

rules

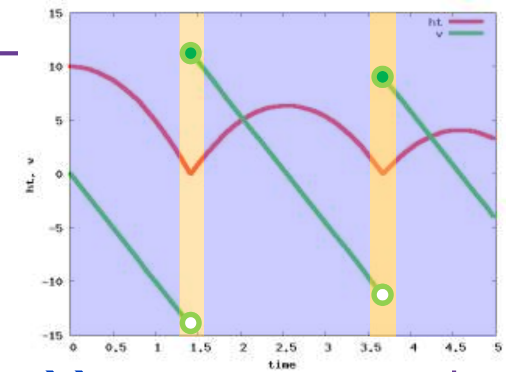
guard

priority

- Describes properties at time 0.
- Family of sawtooth functions with the slope 1 and the range [0, 1)
- Value of f at a specific time point is just known to be [0, 1), but all trajectories reach all values in [0, 1) and oscillate.



Example 2 : Bouncing ball



INIT $\Leftrightarrow$ $ht = 10 \wedge ht' = 0$.

PARAMS $\Leftrightarrow$ $\Box(g = 9.8 \wedge c = 0.5)$.

FALL $\Leftrightarrow$ $\Box(ht'' = -g)$.

BOUNCE $\Leftrightarrow$ $\Box(ht- = 0 \Rightarrow ht' = -c \times (ht'-))$.

INIT, PARAMS, (FALL << BOUNCE).

- When the ball is not on the ground, {INIT, PARAMS, FALL, BOUNCE} is maximally consistent.
- When the ball is on the ground, {INIT, PARAMS, BOUNCE} is maximally consistent.
- At each time point, HydLa adopts a maximally consistent set of rules that respects constraint priority.

Demo

- HyLaGI (HydLa Guaranteed Implementation) and webHydLa
  - http://webhydla.ueda.info.waseda.ac.jp/
  - http://www.ueda.info.waseda.ac.jp/hydla/

Constraint hierarchy

- Constraint hierarchy specified by “<<“ determines possible combination of rules

  INIT, PARAMS, (FALL << BOUNCE)

  where rules with highest priority are “required” constraints

- Basic HydLa (next slide) considers a partially ordered set of “set of rules” induced from the constraint hierarchy.

  {INIT, PARAMS, FALL, BOUNCE}

  {INIT, PARAMS, BOUNCE}

Syntax of Basic HydLa

(program)	P	::=	(RS, DS)
(rule sets)	RS	::=	poset of sets of R
(definitions)	DS	::=	set of D's with different LHSs (= function from R to C)
(definition)	D	::=	$R \Leftrightarrow C$
(constraint)	C	::=	$A \mid C \wedge C \mid G \Rightarrow C \mid \Box C \mid \exists x \,.\, C$
(guard)	G	::=	$A \mid G \wedge G$
(atomic constraint)	A	::=	E *relop* E
(expression)	E	::=	*ordinary expression* $\mid E' \mid E-$

Syntax of Basic HydLa: Comments

- A program is a pair of
  - partially ordered set of "sets of rules" (*RS*) and
  - rule definitions (*DS*).

  Example of *RS*:

  {INIT, PARAMS, BOUNCE} $\prec$ {INIT, PARAMS, FALL, BOUNCE}

  - How to derive *RS* from << is beyond Basic HydLa.
- HydLa / Basic HydLa is a language scheme in which the underlying constraint system is left unspecified.
- ∃x . C realizes dynamic creation of variables.

  Example: creation and activation of new timers

  - ∃ is eliminated at runtime using Skolem functions.

Semantics of Basic HydLa

- Declarative semantics (Ueda, Hosobe, Ishii, 2011)
  - What trajectories does a HydLa program denote?
- Operational semantics (Shibuya, Takata, Ueda, Hosobe, 2011)
  - How to compute the trajectories of a given HydLa program?
- Unlike many other programming languages, declarative semantics was designed first, since
  - completeness of the operational semantics can't be expected and
  - diverse execution methods are to be explored.

Declarative semantics of Basic HydLa

- The purpose of a HydLa program is to define the constraints on a family of trajectories.

$$\overline{x}(t) = \{x_i(t)\}_{i \geq 1} \ \ (t \geq 0)$$

- Declarative semantics, first attempt

$$\overline{x}(t) \vDash (RS, DS)$$

  - Works fine for programs not containing $\square$ in the consequents of conditional constraints $G \Rightarrow C$ [JSSST ’08].

    Example: systems with a fixed number of components and without delays

Declarative semantics of Basic HydLa

- Not only trajectories, but also *effective* constraint sets defining the trajectories, change over time.
  - Reason 1: Maximally consistent sets may change.
  - Reason 2: Conditional constraints may discharge their consequents.
    - When the consequent of a constraint starts with $\square$, whether it's in effect or not depends on whether the corresponding guard held in the past
- Declarative semantics (refined)

$$\langle \overline{x}, Q \rangle \vDash (RS, DS)$$

$Q(t)$: rule definitions with dynamically added consequents

Preliminary: □-closure

- We identify a conjunction of constraints with a set of constraints.
- We regard a set of constraints as a function over time.
  - A constraint C in a program is regarded as a function

$$\begin{cases} C(0) = C, \\ C(t) = \{\ \}\ (t>0)\ . \end{cases}$$

- □-closure * : Unfolds (or *unboxes*) the topmost □-formulas dynamically and recursively.

Example: $C = \{f=0, \Box\{f'=1\}\}$

$$\begin{cases} C^*(0) = \{f=0,\ f'=1,\ \Box\{f'=1\}\} \\ C^*(t) = \{f'=1\}\ \ (t>0) \end{cases}$$

Declarative semantics

$\langle \overline{x}, Q \rangle \vDash (RS, DS) \Leftrightarrow$ (i) $\wedge$ (ii) $\wedge$ (iii) $\wedge$ (iv) , where

(i) $\forall t \forall R (Q(R)(t) = Q(R)^*(t))$ ☐-closure

(ii) $\forall t \forall R (DS^*(R)(t) \subseteq Q(R)^*(t))$ extensiveness

(iii) $\forall t \exists E \in RS$ (

$(\overline{x}(t) \Rightarrow \{Q(R)(t) \mid R \in E\})$ satisfiability

$\wedge \neg \exists \overline{x}' \exists E' \in RS$ (
$\forall t' < t \, (\overline{x}'(t') = \overline{x}(t'))$
$\wedge E \prec E'$
$\wedge \overline{x}'(t) \Rightarrow \{Q(R)(t) \mid R \in E'\})$ maximality

$\wedge \forall d \forall e \forall R \in E$ (
$(\overline{x}(\mathrm{t}) \Rightarrow d) \wedge ((d \Rightarrow e) \in Q(R)(t))$
$\Rightarrow e \subseteq Q(R)(t)))$ $\Rightarrow$-closure

(iv) $Q(R)(t)$ at each t is the smallest set satisfying (i)-(iii)

Example 3 : Absence of back propagation

$$P = ((\wp(\{D,E,F\}), \subsetneq), DS)$$

$$\begin{aligned} DS = \{\ & D \Leftrightarrow y = 0, \\ & E \Leftrightarrow \Box(y' = 1 \wedge x' = 0), \\ & F \Leftrightarrow \Box(y = 5 \Rightarrow x = 1)\ \} \end{aligned}$$

a. $y(t) = t,\ \ x(t) = 1$ satisfies D, E, F at $0 \le t$.

b. $y(t) = t,\ \ x(t) = 2$ satisfies D, E, F at $0 \le t < 5$ and D, E at $t = 5$. It again satisfies D, E, F at $t \ge 5$.

c. $y(t) = t,\ \ x(t) = 2\ (t < 5),\ \ x(t) = 1\ (t \ge 5)$ satisfies D, E, F at $0 \le t < 5$ and D, F at $t = 5$. It again satisfies D, E, F at $t \ge 5$.

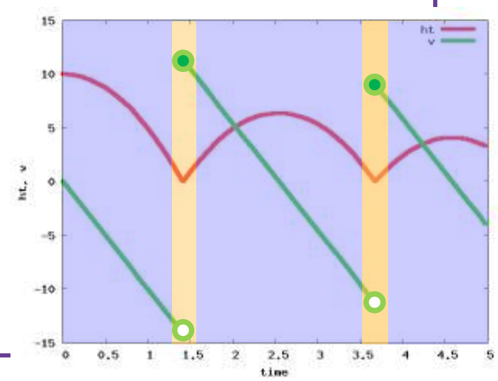
All of a., b. and c. satisfy local maximality and hence satisfy P.

Example 4 : Bouncing Ball, revisited

$P \;=(RS,\ DS)$

$RS=(\{\{I,C,B\},\{I,C,F,B\}\},\{\{I,C,B\} \prec \{I,C,F,B\}\})$

$$
\begin{aligned}
DS=\{\ & I \;\Leftrightarrow ht=10 \wedge ht'=0,\\
& C \;\Leftrightarrow \Box(g=9.8 \wedge c=0.5),\\
& F \;\Leftrightarrow \Box(ht''= -g),\\
& B \;\Leftrightarrow \Box(ht-=0 \Rightarrow ht'=-c\times(ht'-))\}
\end{aligned}
$$



- ht and ht' are not differentiable when bouncing
- However, to solve ODEs on ht and ht', *right continuity* of ht and ht' at the bouncing must be assumed
- To determine ht at the bouncing, *left continuity* of ht must be assumed as well. (cf. ht' is determined from B.)

➔ Trajectories with differential constraints should assume both right and left continuity with appropriate priority.

Example 5 : Behaviors defined without ODEs

$$P = (RS, DS)$$

$$RS = (\{\{A,C\}, \{A,B,C\}\}, \{\{A,C\} \prec \{A,B,C\}\})$$

$$\begin{aligned} DS = \{ & A \Leftrightarrow f=0 \wedge \Box(f' = 1), \\ & B \Leftrightarrow \Box(g=0), \\ & C \Leftrightarrow \Box(f=5 \Rightarrow \exists a.(a=0 \wedge \Box(a'=1) \\ & \qquad \wedge \Box(a=2 \Rightarrow g=1))) \} \end{aligned}$$

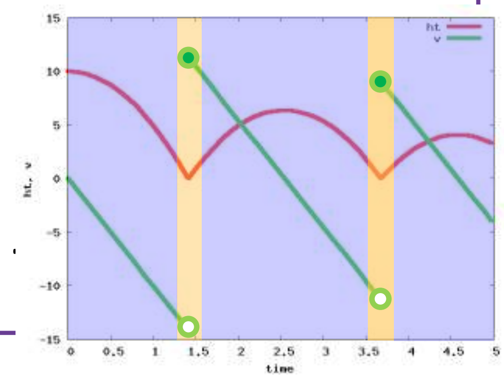
- g is an impulse function that fires at time 7 (= 5+2).
  - an example of non-right-continuous functions

$$\Box(0.9<a \wedge a<1.1) \wedge \Box(a'=b)$$

- a is a set of all smooth trajectories with the range (0.9, 1.1) . Could be used for specification but not for modeling.

Example 6 : Zeno behavior

$P = (RS, DS)$
$RS = (\{\{I,Pa,B\},\{I,Pa,F,B\}\},\{\{I,Pa,B\} \prec \{I,Pa,F,B\}\})$
$DS = \{\ I \Leftrightarrow ht=10 \wedge ht'=0,$
$Pa \Leftrightarrow \Box(g=9.8 \wedge c=0.5),$
$F \Leftrightarrow \Box(ht'' = -g),$
$B \Leftrightarrow \Box(ht- = 0 \Rightarrow ht' = -c\times(ht'-))\}$



- This doesn't define a trajectory *after* the Zeno time.
- A rule for defining the trajectory after Zeno:

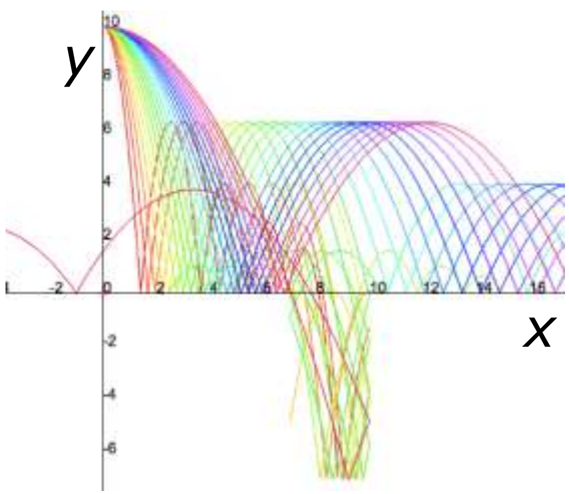
$\Box(ht- = 0 \wedge ht'- = 0 \Rightarrow \Box(ht=0))$

  - Checking of the guard condition would require a technique not covered by the current operational semantics.

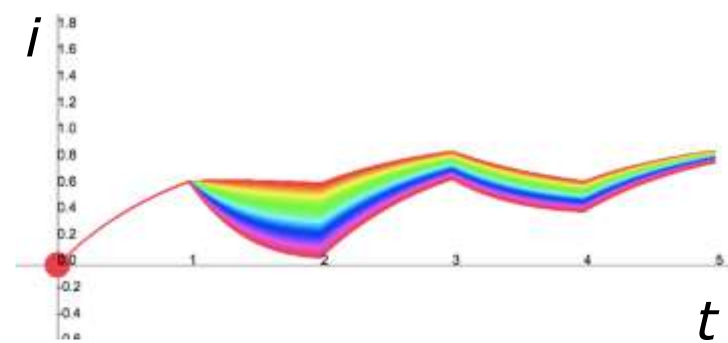
Execution algorithm and implementation

HyLaGI: A symbolic simulator

- C++ (frontend) and Mathematica (backend), 27kLOC
- KV library[1] for interval computation
- Optimized computation by exploiting the locality of constraints
- webHydLa[2] for visualization



Bouncing ball on a ground with a hole



Electrical circuit

[1] http://verifiedby.me/

[2] http://webhydla.ueda.info.waseda.ac.jp/

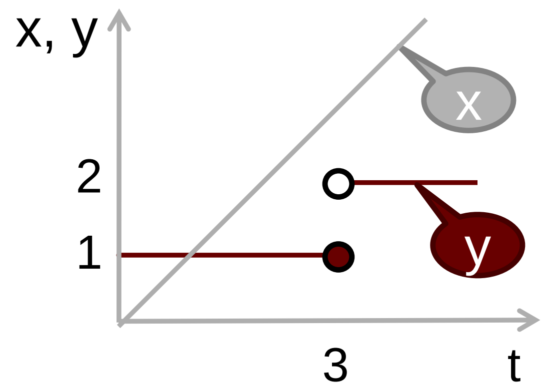
Rigorous tools for hybrid systems

Tool	Approach
Acumen	Validated Numerical Simulation
Flow*	Taylor model + Domain contraction
dReach/dReal	Interval Constraint Propagation + Bounded Model Checking with Unrolling + SMT Solving
SpaceEx	Template Polyhedra & Support functions
KeYmaera & KeYmaera X	Symbolic Theorem Prover based on differential invariants
HyLaGI	Symbolic + Affine Arithmetic + Interval Newton method

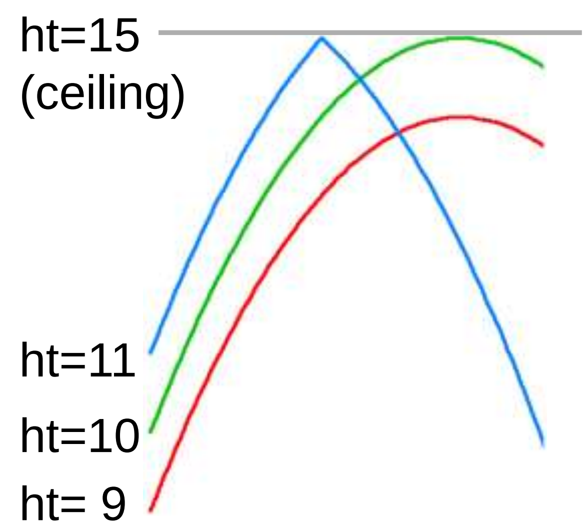
Execution algorithm of HydLa should handle:

1. conditions that starts to hold "after" some time point
   - need to compute the greatest lower bound of a time interval

A ⇔ x=0.
B ⇔ □ (y=1).
C ⇔ □ (x'=1 ∧ (x>3 ⇒ y=2)).
A, (B << C).



2. initial values given as intervals
   - could be divided into a subinterval that entails a guard and another that does not entail the guard
3. systems with symbolic parameters
   - needs symbolic computation



Operational semantics

- For simulation, we need to consider a class of "computable" trajectories.
- Computable trajectories: those that have possibly *parametric* equational closed forms
  - ODEs without closed-form solutions are to be over-approximated by parametric equational closed forms.

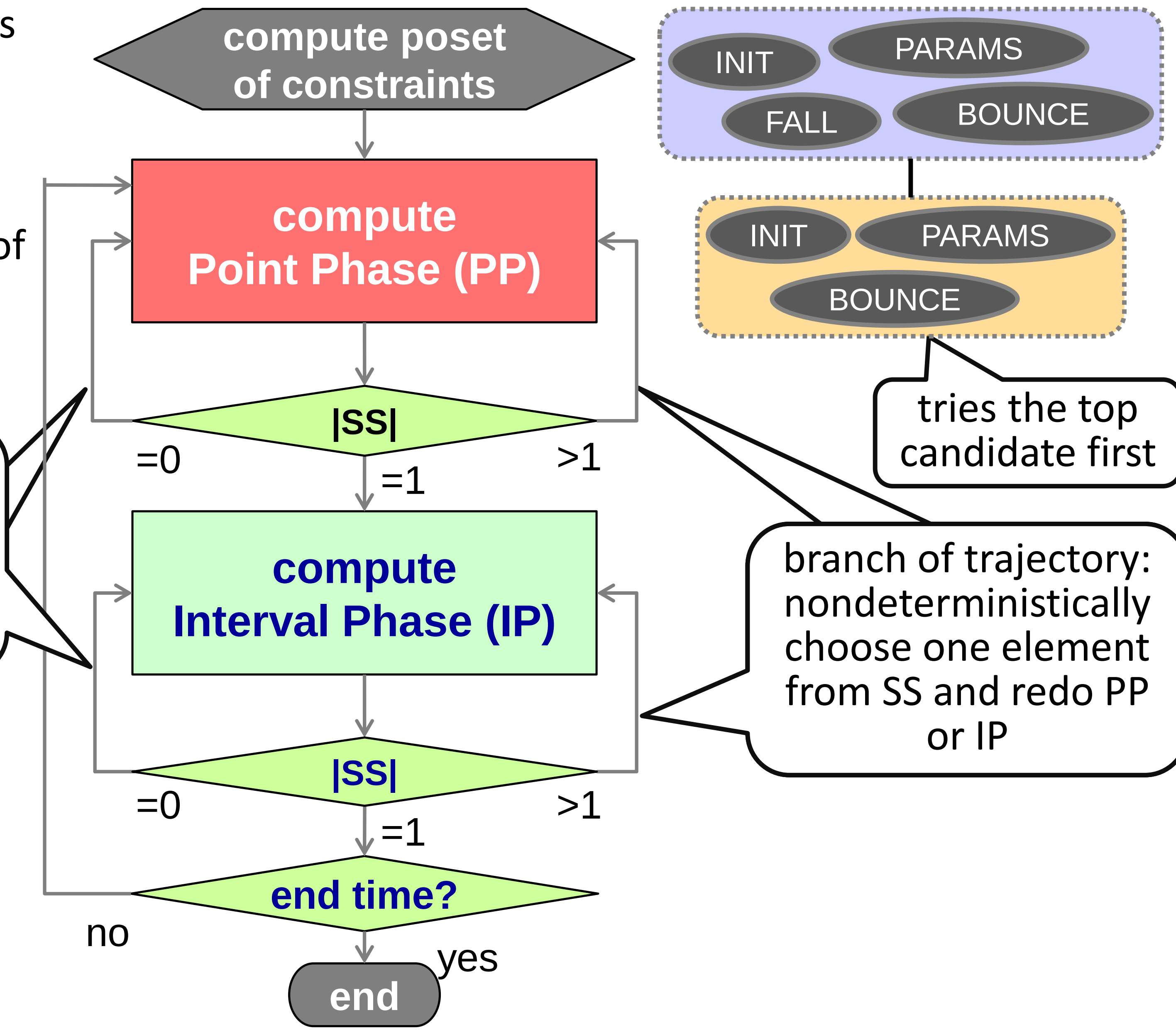
Execution algorithm

each phase updates the maximal consistent set and simulation time T

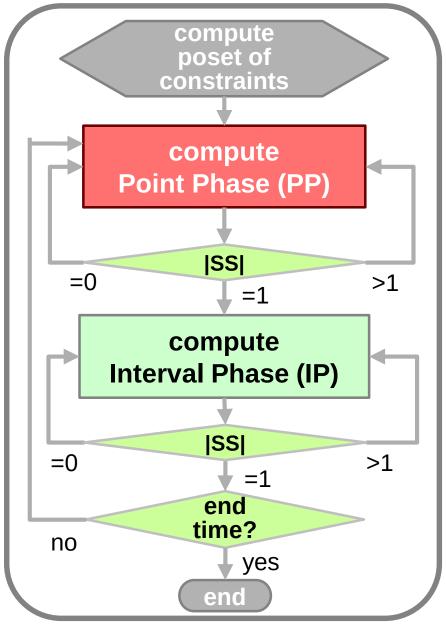
SS (store set) : set of possible stores

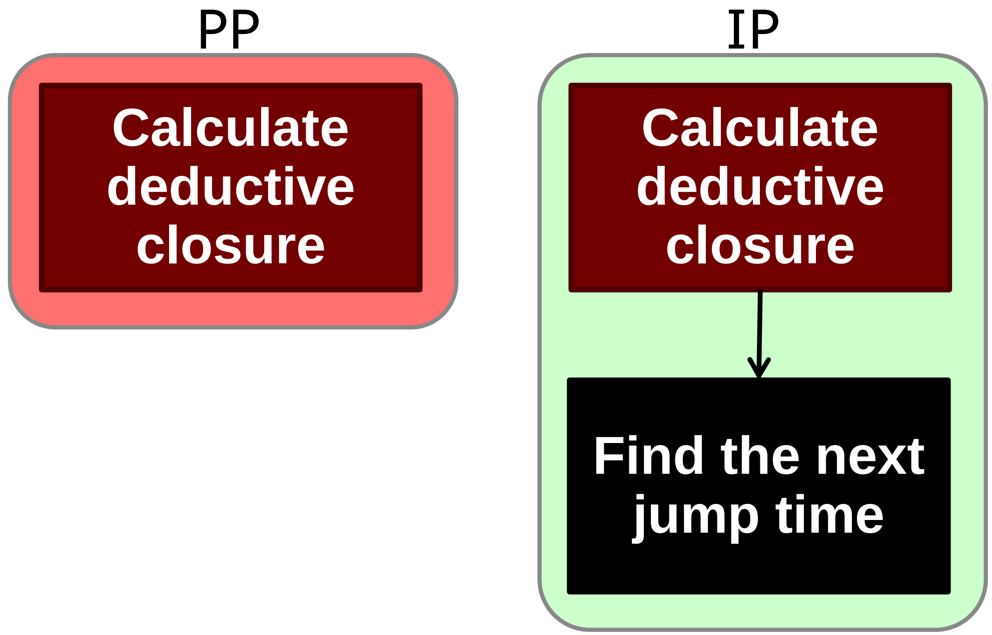
failure: choose the next candidate set and redo PP or IP

an element of SS represents a result of execution of PP or IP



Algorithm for Point Phase and Interval Phase





Closure calculation repeatedly checks the antecedents of conditional constraints

IP computes the next jump time (minimum of the following):

1. a conditional constraint becomes effective
2. a conditional constraint becomes ineffective
3. a ruled-out constraint becomes consistent with effective ones
4. the set of effective constraints becomes inconsistent

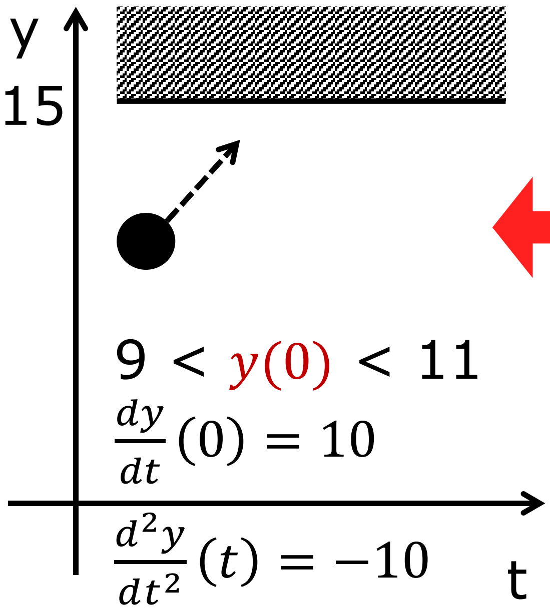
Where's nondeterminism?

- Choice of *maximally* consistent set of rules
- Calculating deductive closure
  - Guard $(g \Rightarrow \cdots)$ may hold or may not hold depending on parameter values
    (e.g., will the thrown ball reach the wall?)
  - We calculate a "strengthened" constraint store for each case
- Finding the next possible jumps time
  - Reason of the next jump may depend on parameter values
    (e.g., will the ball hit the wall or the floor first?)
  - Together with each jump time, calculate a strengthened constraint store which causes that jump first

Example: Bouncing ball with ceiling

- Thrown towards ceiling from some unknown height

y': dy/dt
y–: left limit of y
□: ∀t ≥ 0



$9 < y(0) < 11$

$\frac{dy}{dt}(0) = 10$

$\frac{d^2y}{dt^2}(t) = -10$

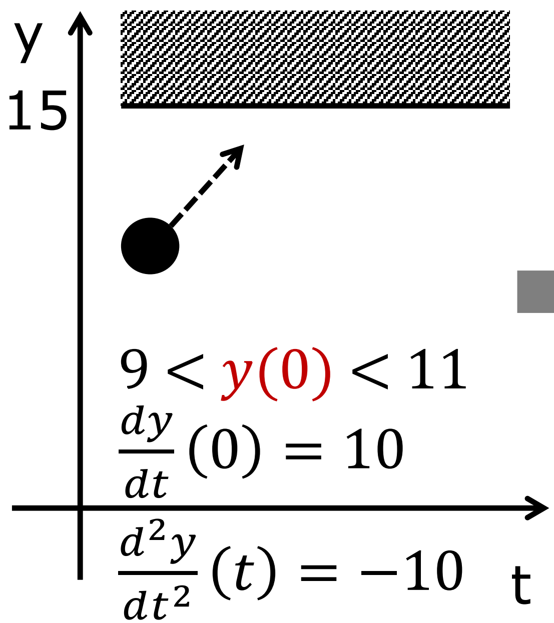
HydLa

INIT ⇔ 9 ≤ y ≤ 11 ∧ y' = 10.
FALL ⇔ □(y'' = –10).
BOUNCE ⇔ □(y– = 15 ⇒
y' = –(4/5) * y'–).

INIT, (FALL << BOUNCE).

Symbolic execution of HydLa models

- Use symbolic parameters to handle uncertainties
- Includes ODE solving, Quantifier Elimination (for consistency checking and case splitting), optimization problem (for computing time of discrete change)



$$9 < y(0) < 11$$
$$\frac{dy}{dt}(0) = 10$$
$$\frac{d^2y}{dt^2}(t) = -10$$

Case 1 (fall)

$9 < y(0) < 10$

y: $10t - 5t^2 + y(0)$

Case 2 (touch)

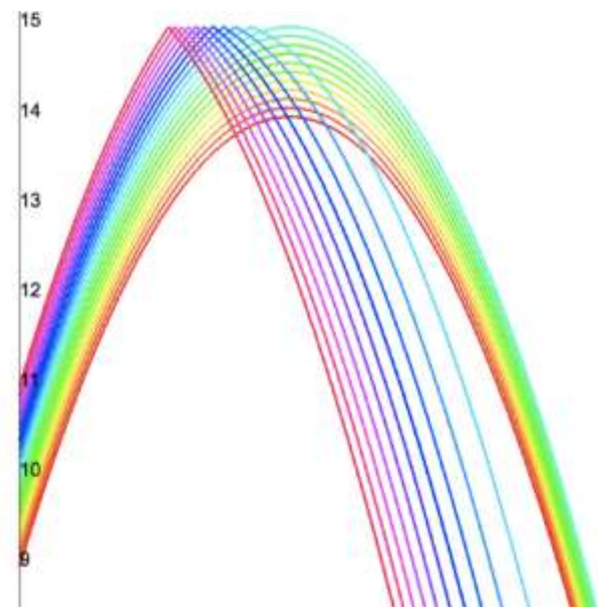
$y(0) = 10$

y: $10t - 5t^2 + 10$

Case 3 (collide)

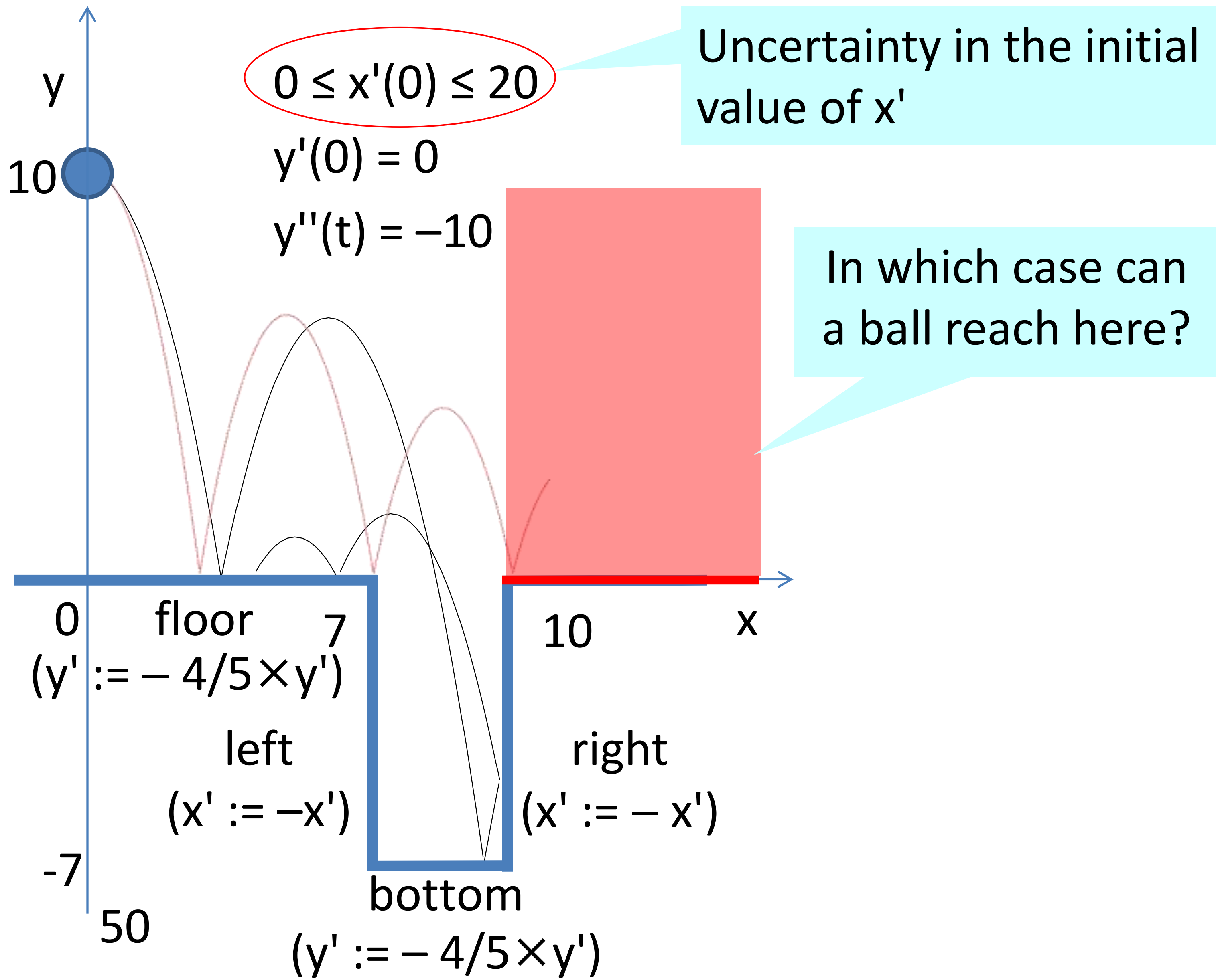
$10 < y(0) < 11$

y: $15 - \sqrt{10y(0)}...$



Result plots

Bouncing ball on a ground with a hole



Bouncing ball on a ground with a hole

```
INIT      <=> y = 10  ∧  y' = 0  ∧  x = 0  ∧  0 ≤ x' ≤ 20.
FALL      <=> □(y'' = –10).
BOUNCE    <=> □(y– = – 7  ∨  (x– ≤ 7 ∨ x– ≥ 10) ∧ y– = 0
                    =>  y' = –(4/5) * y'– ).
XCONST    <=> □(x'' = 0).
XBOUNCE   <=> □((x– = 7 ∨ x– = 10) ∧ y– < 0  =>  x' = –x'–).

INIT, (FALL << BOUNCE), (XCONST << XBOUNCE).
ASSERT( ! (y ≥ 0 ∧ x ≥ 10)).
```

Search when the ball reaches the goal zone

Bouncing ball on a ground with a hole

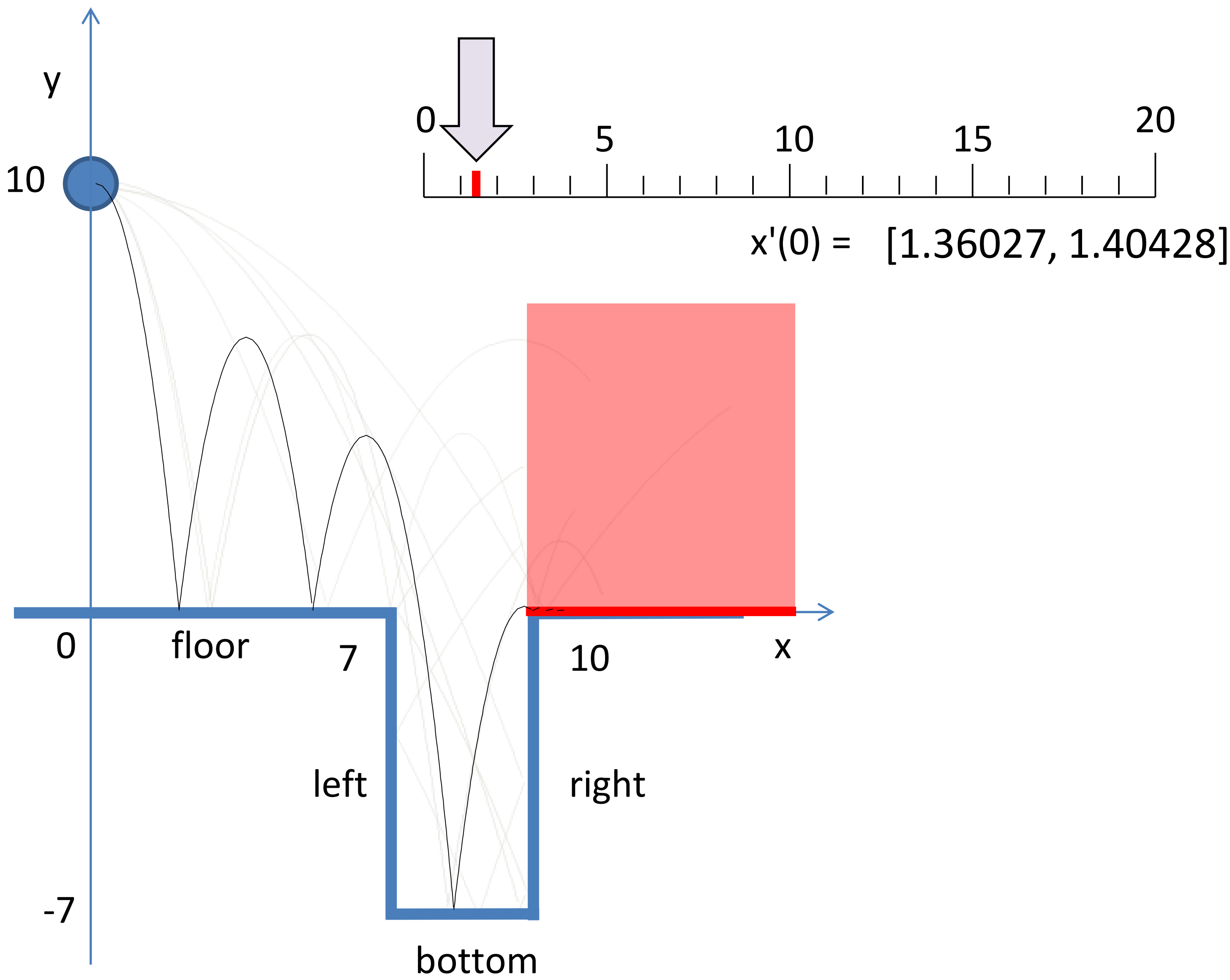
```
INIT <=> y = 10 ∧ y' = 0 ∧ x = 0 ∧ 0 ≦ x' ≦ 20.
FALL <=> □(y'' = -10).
BOUNCE <=> □(y- = -7 | (x- ∧ 7 | x- ≧ 10)
                    ∧ y- = 0 => y' = -(4/5) * y'-)
XC
XE

INIT, FALL << BOUNCE, XCONST << XBOUNCE.
ASSERT(!(y ≧ 0 ∧ x ≧ 10)).
```

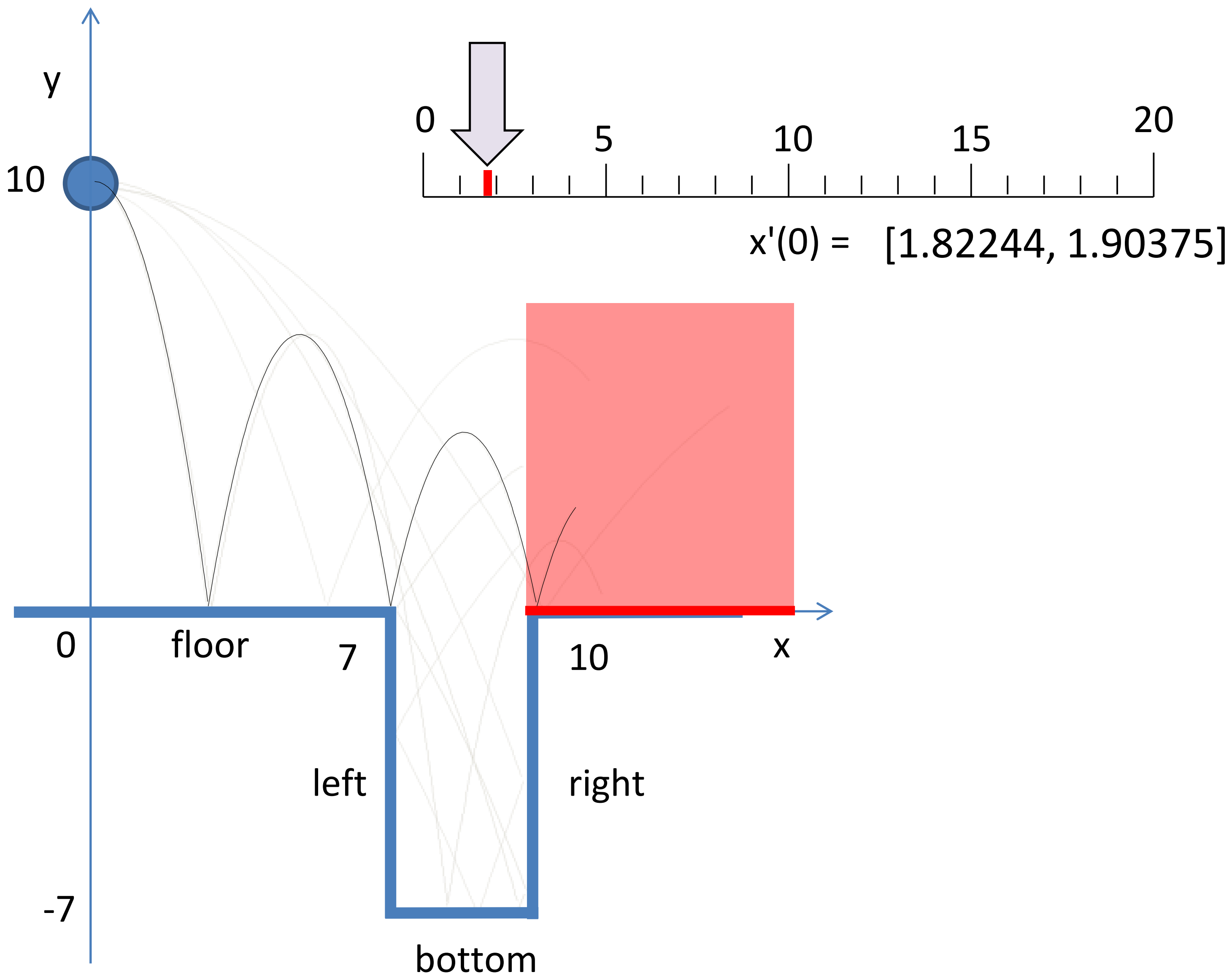
Successfully simulated with automatic case analysis
(50 cases including unreachable ones)
(up to 20 seconds, six discrete changes)

Search when the ball reaches the goal zone

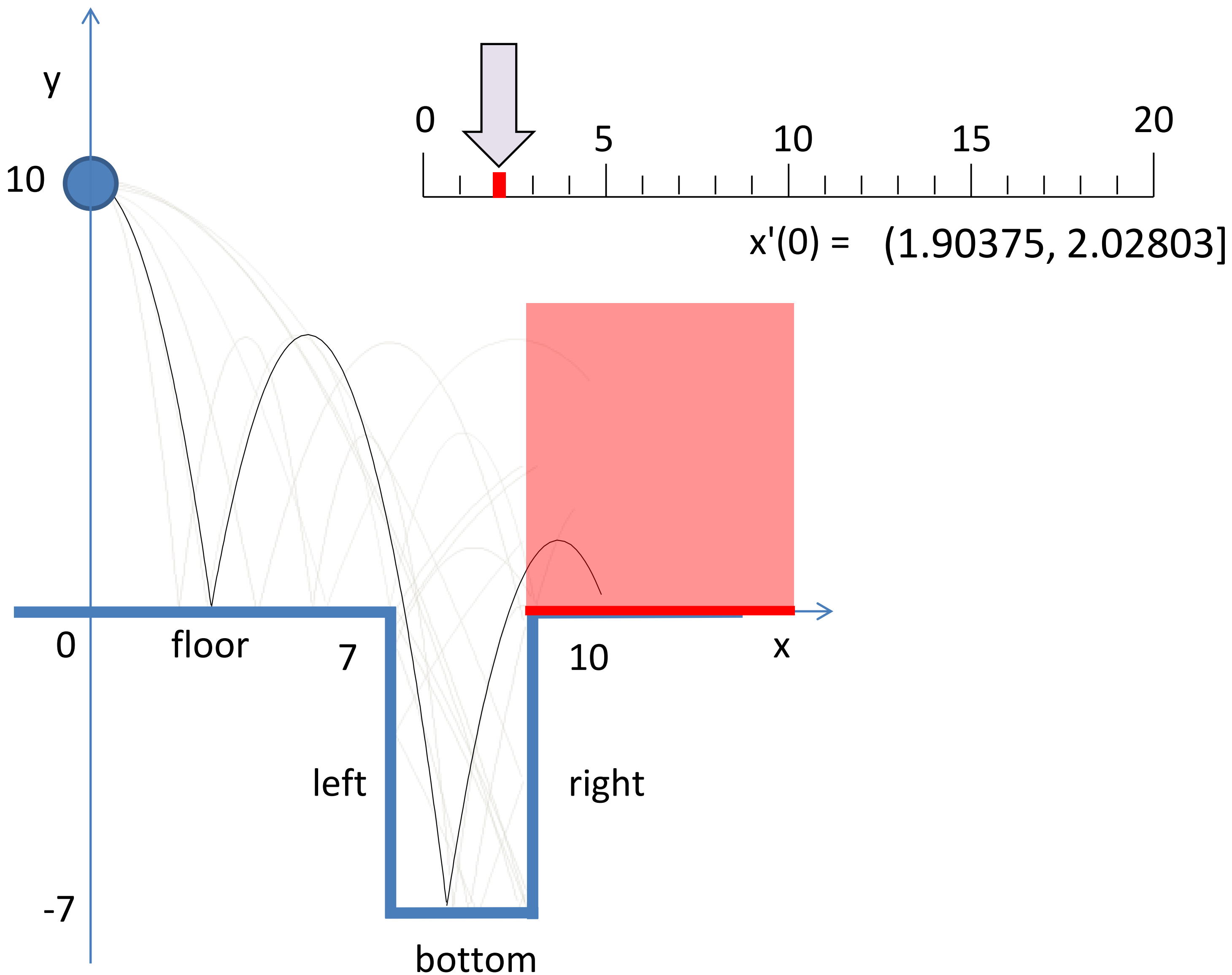
Bouncing ball on a ground with a hole (1/9)



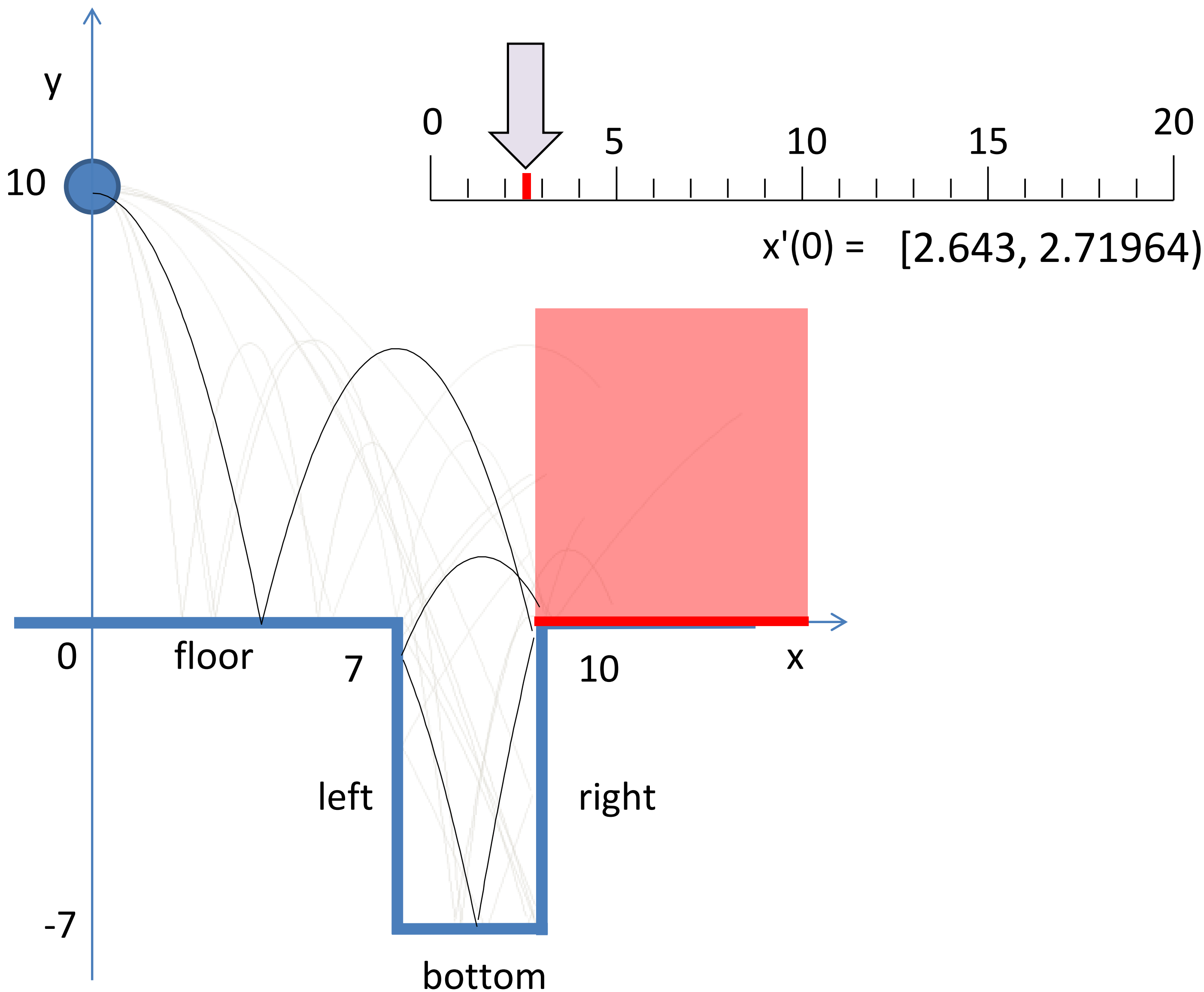
Bouncing ball on a ground with a hole (2/9)



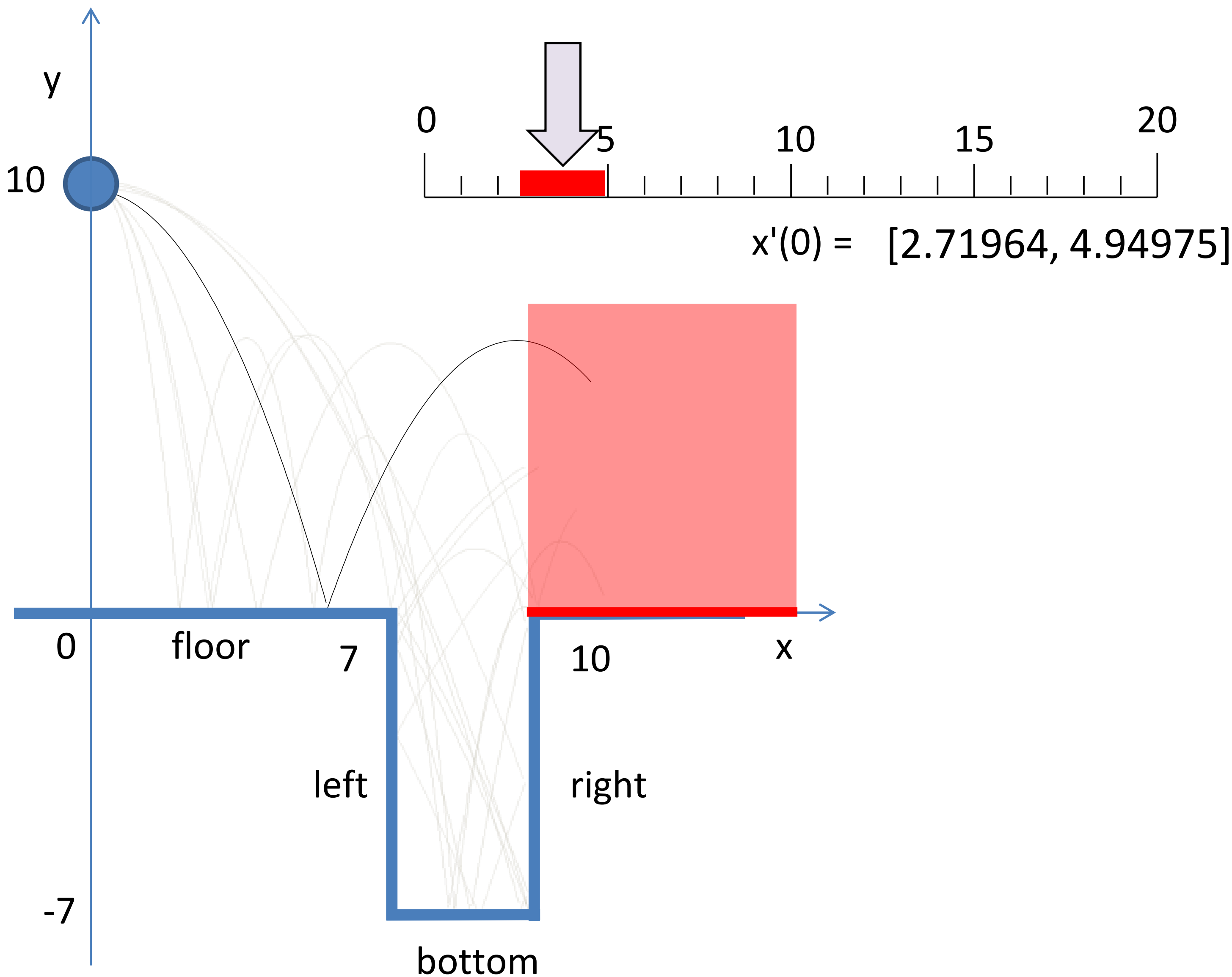
Bouncing ball on a ground with a hole (3/9)



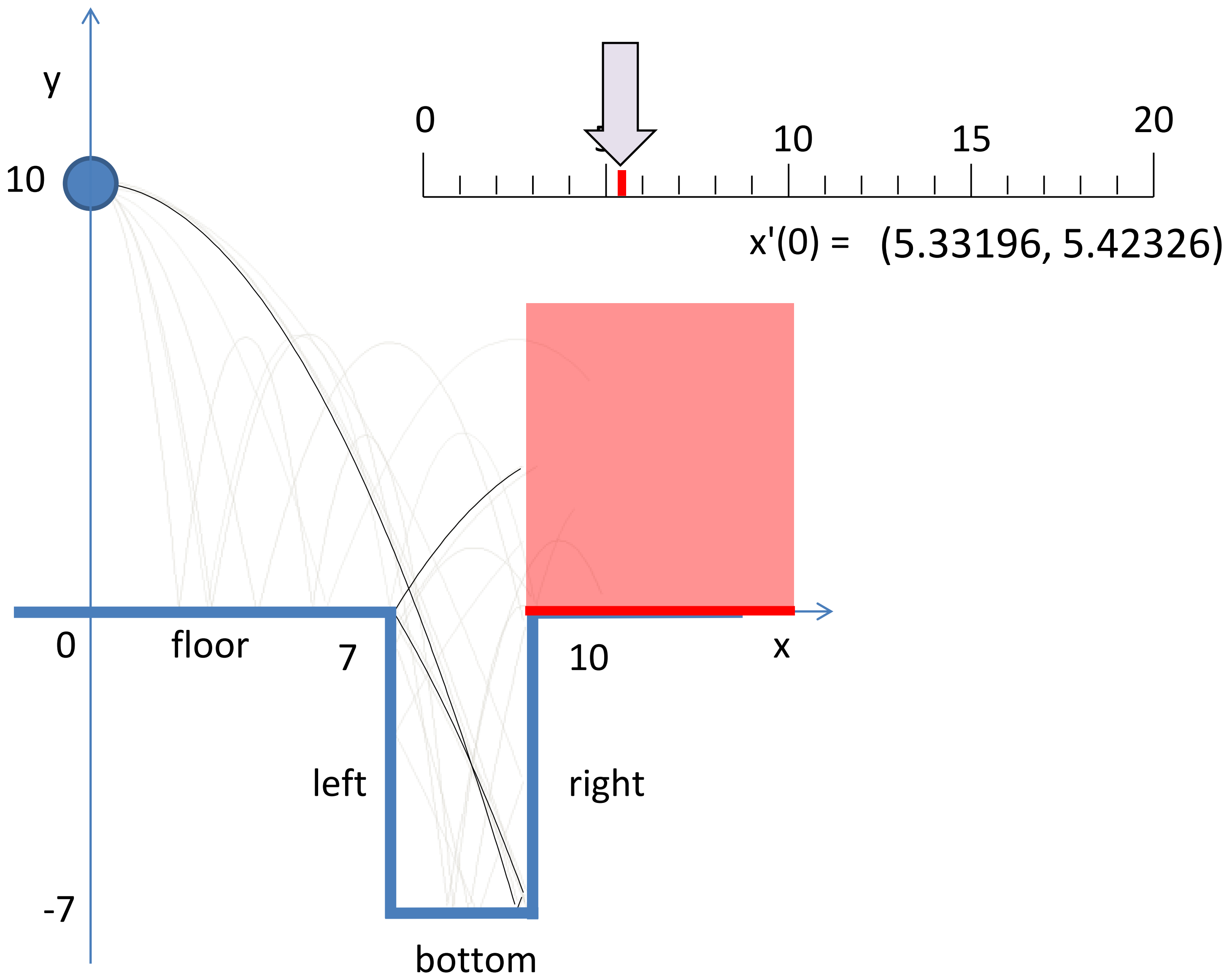
Bouncing ball on a ground with a hole (4/9)



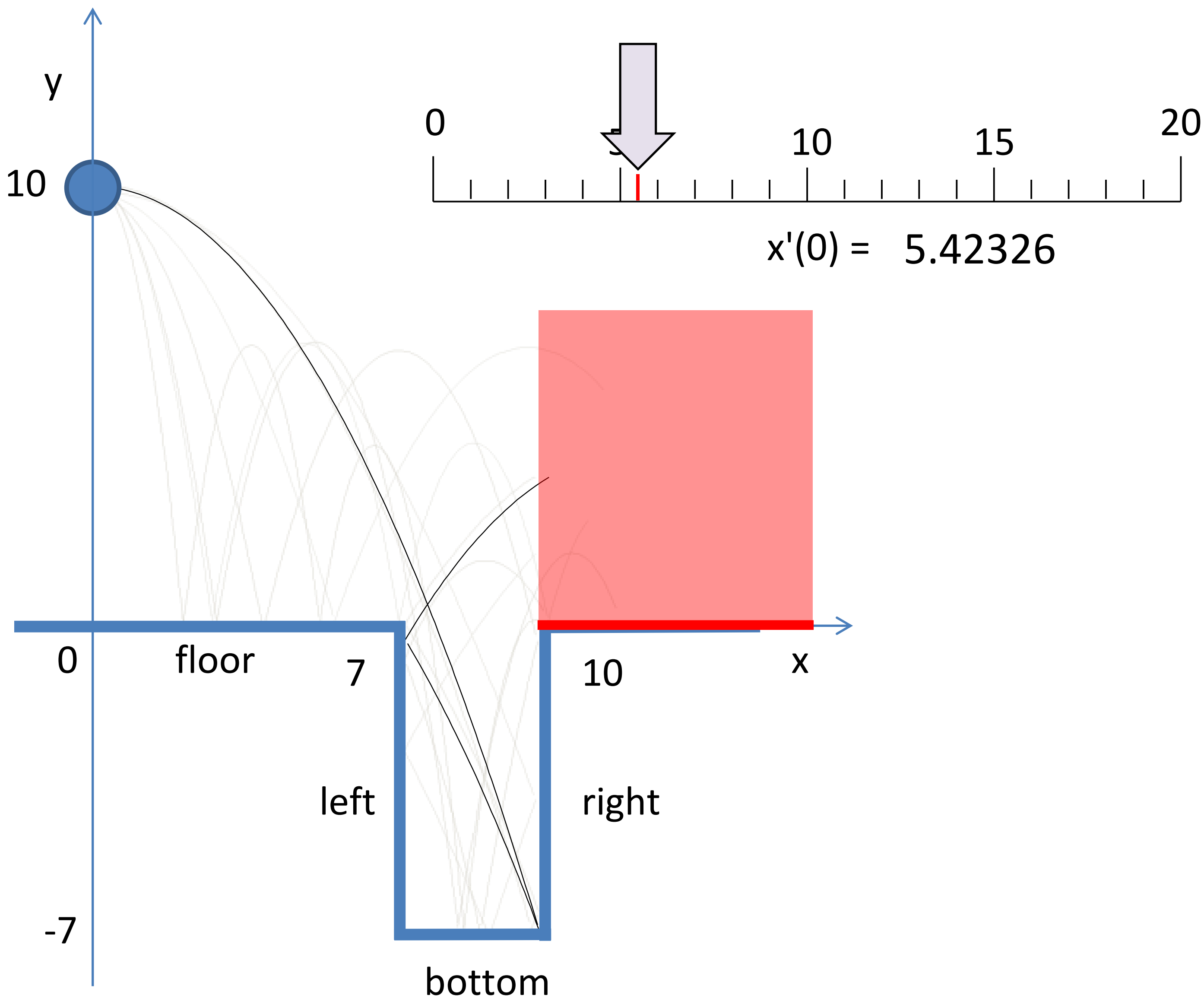
Bouncing ball on a ground with a hole (5/9)



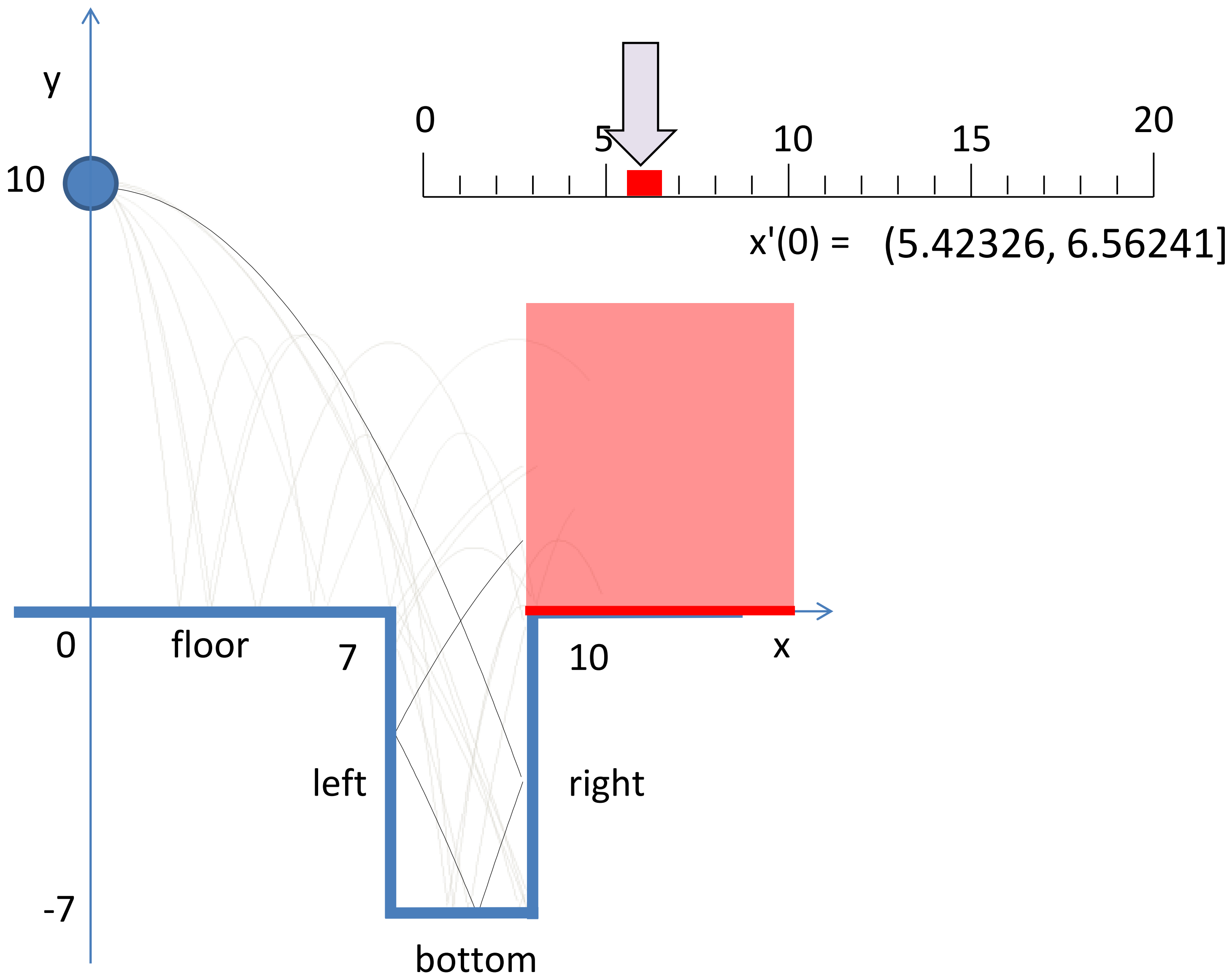
Bouncing ball on a ground with a hole (6/9)



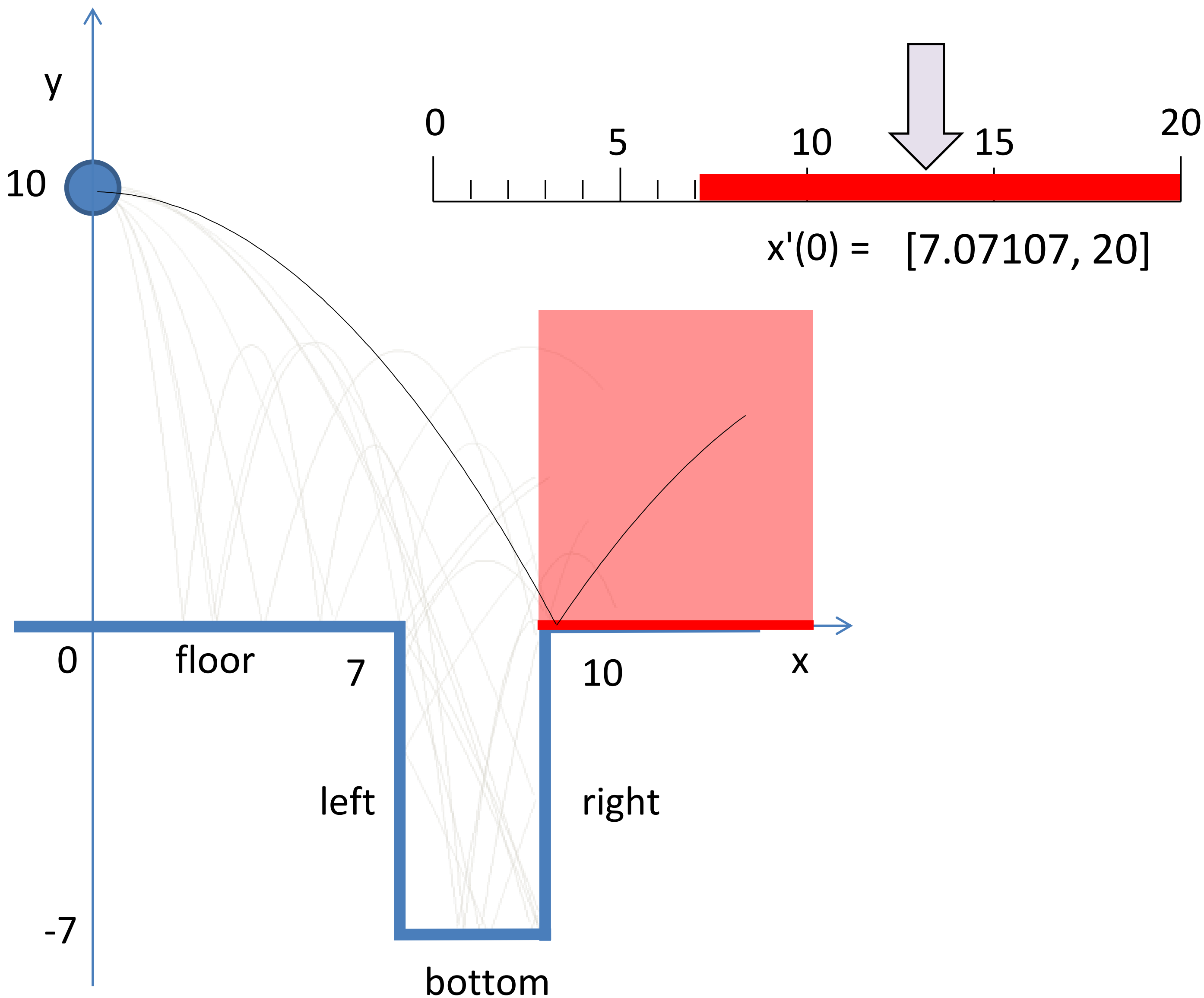
Bouncing ball on a ground with a hole (7/9)



Bouncing ball on a ground with a hole (8/9)



Bouncing ball on a ground with a hole (9/9)

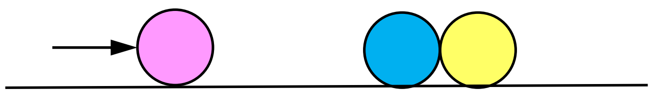


Instantaneous events

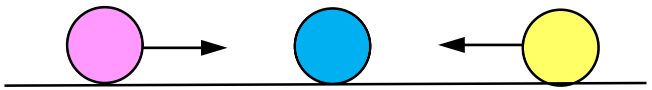
- Hybrid systems handle discrete events
  - as abstraction of quick physical change (e.g., collision)
  - to represent computational aspects (e.g., controller)
- Superdense time allows multiple events at the same time
  - (t, n)
    - t: real
    - $n = 0, 1, 2, \ldots$: event number at time t
- In our constraint-based framework, what can we do with the standard notion of time?

Modeling behaviors with symbolic purturbation

- Simultaneous collision

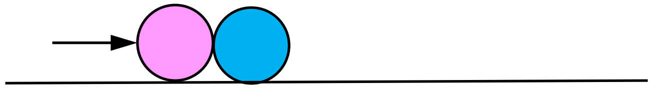


[1], Fig.8-9



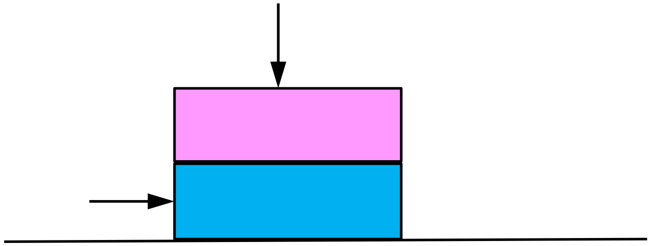
[1], Fig.11-12 (equal mass)
[1], Fig.14 (different mass)

- Collision
  \+ pushing at $1 \leq t \leq 3.5$



[1], Fig.15

- Masses with friction



[1], Fig.28

[1] Edward Lee, Constructive Models of Discrete and Continuous Physical Phenomena, *IEEE Access*, Vol.2, 2014

Representing computational aspects

- Solution 1: Form a network of constraints

```
N := {n0 .. n5}.
F := {f0 .. f5}.
[](f0 = 1 & n0 = n & f = f5).

n = 3.
{ [](N[i] > 0  => F[i+1] = F[i] * N[i] & N[i+1] = N[i] - 1),
  [](N[i] <= 0 => F[i+1] = F[i] & N[i+1] = N[i])
 | i in {1..|F|-1} }.
```

- Solution 2: use ∃

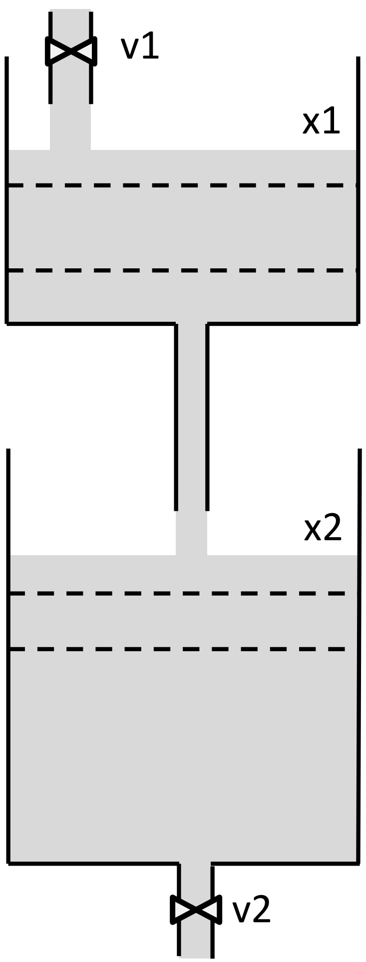
```
F(0, y) <=> y=1.
F(x, y) & x>0 <=> ∃z.(y = n*z & F(x-1, z))
```

Cooperation of symbolic and numeric techniques

Shota Matsumoto and Kazunori Ueda: *Proc. TIME 2016*, pp.4-11, Oct. 2016

Discrete change is often hard(er) to compute

Exmple: water level control



$$\begin{cases} \frac{dx1}{dt} = -x1 + 3 \text{ (v1: open)} \\ \frac{dx1}{dt} = -x1 - 2 \text{ (v1: closed)} \end{cases}$$

1 (v1 → close)

-1 (v1 → open)

$1.9 \leq x1(0) \leq 1.9001$

$x2(0) = 1$

1 (v1 → close & v2 → open)

0 (v2 → close)

$$\begin{cases} \frac{dx2}{dt} = x1 - x2 - 5 \quad \text{(v2: open)} \\ \frac{dx2}{dt} = x1 \text{ (v2: closed)} \end{cases}$$

- First continuous change

$$x2(t) = -\frac{-8 + 7e^{t} - 2t - t * x1(0)}{e^{t}}$$

- Mathematica cannot symbolically solve

$$x2(t) = 0$$

- We need to handle it with interval numerical methods

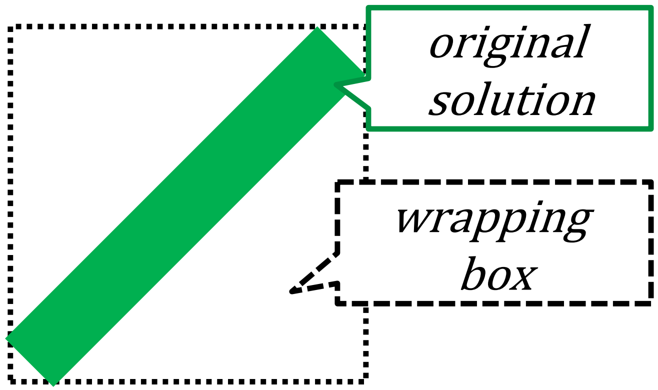
Interval arithmetic

Arithmetic defined on intervals of reals

- e.g. $[a,b]+[c,d]=[a+c,b+d]$
  $[a,b]-[c,d]=[a-d,b-c]$

Shortcoming: explosion of interval width

Cause1: Wrapping Effect



Cause2: Dependency problem

$$X := [-1,1]$$
$$f(x) := x - x$$
$$f(X) = X - X$$
$$= [-1,1] - [-1,1]$$
$$= [-2,2]$$

❯ Solve by handling symbolic parameters

Symbolic vs. numerial methods

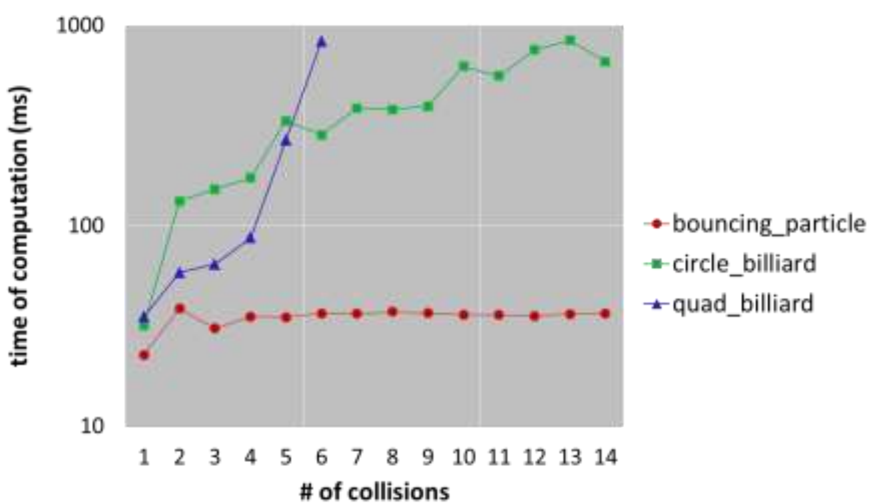
Symbolic

Advantage

Retains parametric info

Disadvantage

Growth of size of math formulae



tradeoff

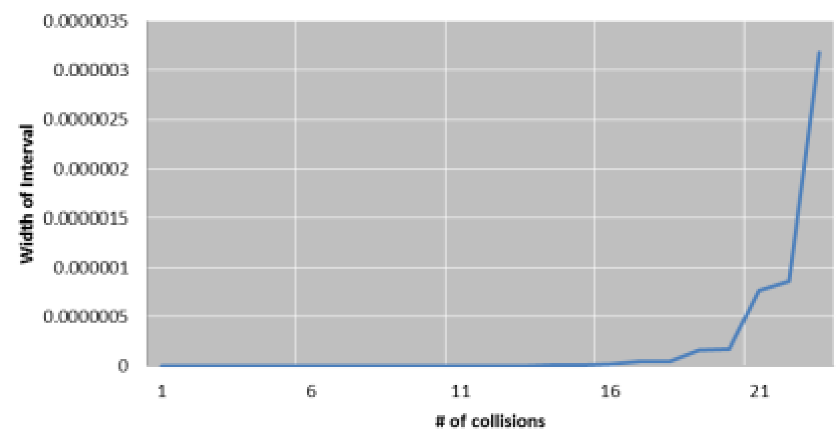
Numerical

Advantage

Handles vast class of models

Disadvantage

Accumulation of errors

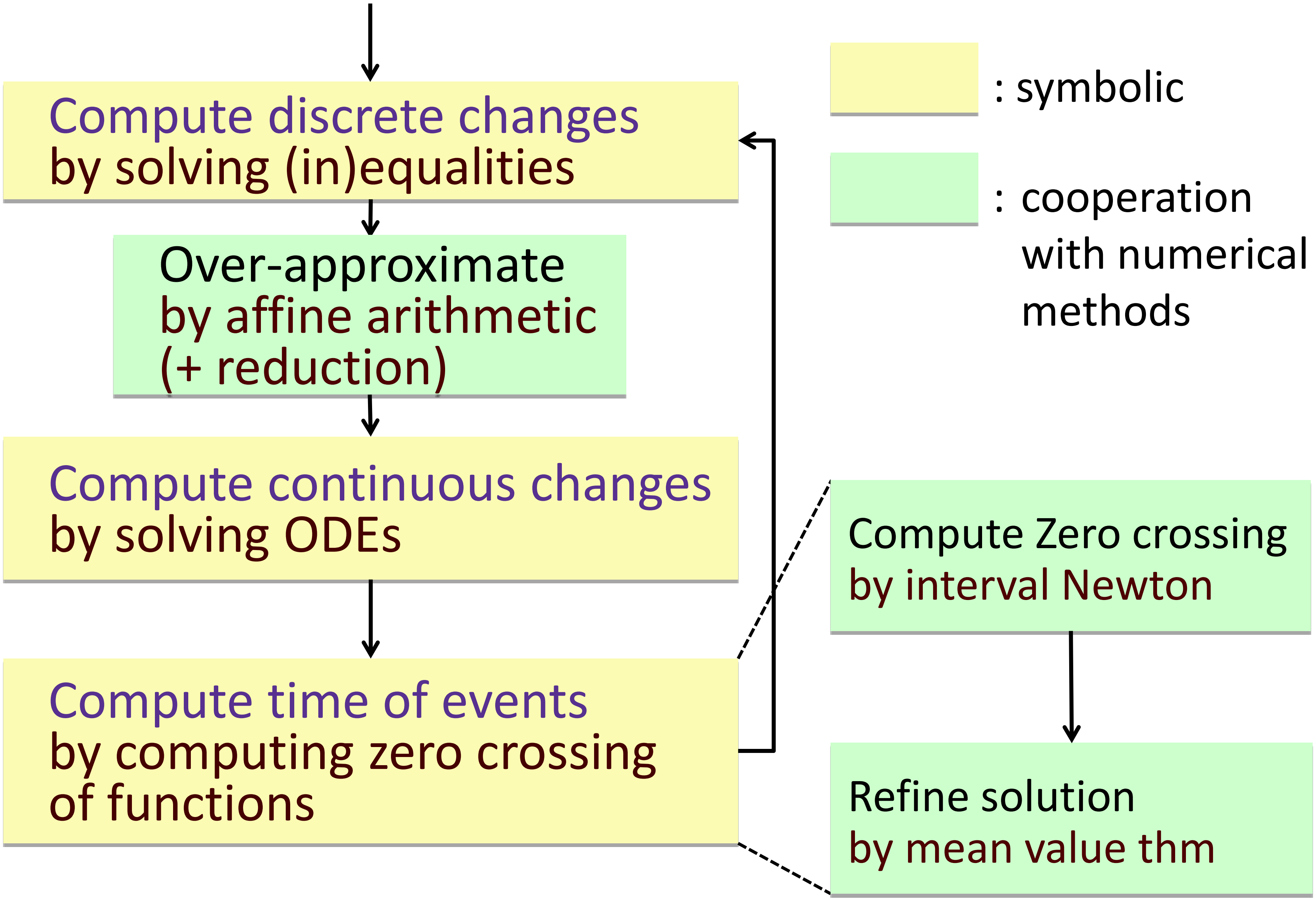


Cooperation of symbolic and numeric methods

- Use affine arithmetic (AA) to approximate complex formulae
  - to reduce computational cost
  - while retaining linear terms of parameters

- Use interval Newton method and mean-value theorem to compute discrete change rigorously
  - to handle systems that are hard to compute symbolically
  - while retaining linear terms of parameters

Cooperation of symbolic and numeric methods



Affine Arithmetic

- Extended version of Interval Arithmetic
  - Expresses uncertainty in affine form

Affine form

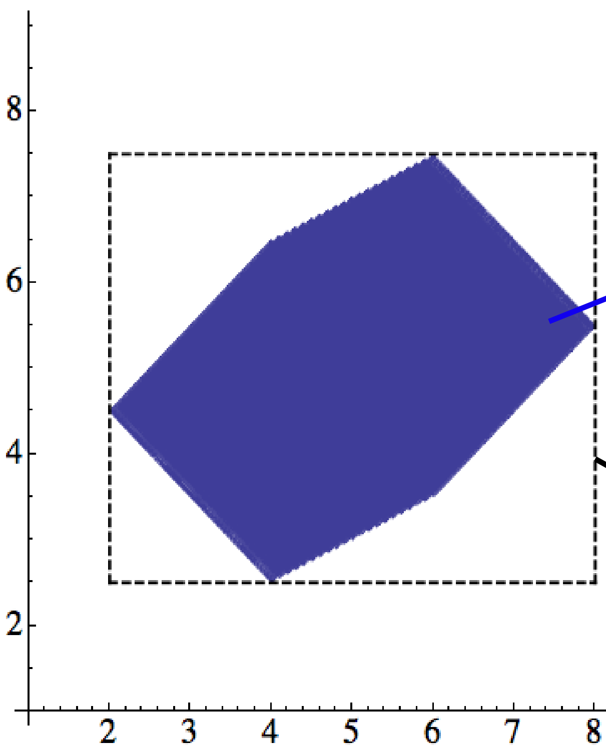
$$\begin{cases} X = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\ -1 \le \varepsilon_i \le 1 \end{cases}$$

  - Each ε_i represents uncertainty just in the same manner as symbolic parameters in symbolic execution
  - Each x_i $(i > 0)$ represents the effect of ε_i, while x_0 represents the center

[1] de Figueiredo, L. H. and Stolfi, J.: *Numerical Algorithm*, 37 (1–4), 147–158, 2004

Affine Arithmetic

- Affine forms represent zonotopes, a polygon with parallel opposite edges
- Symbolic parameters ε_i retain first-order dependencies between uncertain values



$$X = 5 + \varepsilon_1 - \varepsilon_2 + \varepsilon_3$$
$$Y = 5 + \varepsilon_1 + \varepsilon_2 + 0.5\varepsilon_3$$

$$X = [2, 8]$$
$$Y = [2.5, 7.5]$$

Over-approximation by Affine Arithmetic

We use affine arithmetic to over-approximate symbolic formulas

- It reduces computational cost for complex formulas
- Number of preserved parameters can be reduced

Example

$$f(x) \coloneqq (x+1)^2 - 2x$$

$$X \coloneqq 0 + 0.1\,\varepsilon_1 \; (= [-0.1, 0.1])$$

$$\begin{aligned} f(X) &= (1 + 0.1\varepsilon_1)^2 - 0.2\,\varepsilon_1 \\ &= 2(1 + 0.1\varepsilon_1) - 0.995 + 0.005\,\varepsilon_2 - 0.2\,\varepsilon_1 \\ &= 2 + \underline{0.2\varepsilon_1 - 0.2\,\varepsilon_1} - 0.995 + 0.005\,\varepsilon_2 \\ &= 1.005 + 0.005\varepsilon_2 \quad (= [1, 1.01]) \end{aligned}$$

cancelled by preserved dependency

Computation of Event Time

- Goal: compute the solution of $f(t, \vec{p}) = 0$ w.r.t. t that preserves the linear terms of the parameters $\vec{p}$
- Assume that the guard is described by a single equation: $g(\vec{x}) = 0$

Step 1. Substitute solution of ODEs into $g(\vec{x})$ and obtain $f(t, \vec{p})$

Step 2. Solve $f(t, \vec{p}) = 0$ by interval Newton method and obtain solution interval T

Step 3. Obtain linear over-approximation $F(t, \vec{p})$ that encloses $f(t, \vec{p})$ in T using mean value thm

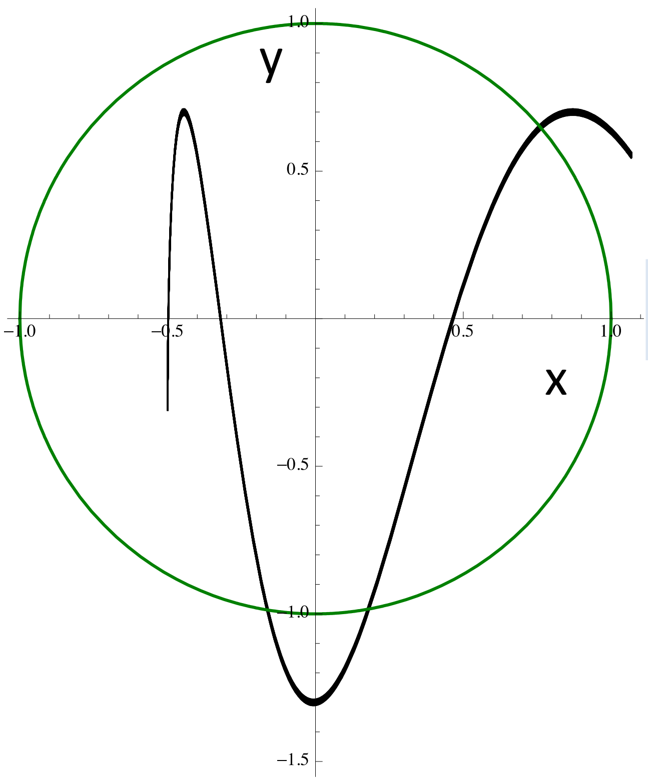
Step 4. Compute zero-crossing of $F(t, \vec{p})$ symbolically

Step 1. Substitution of Trajectory

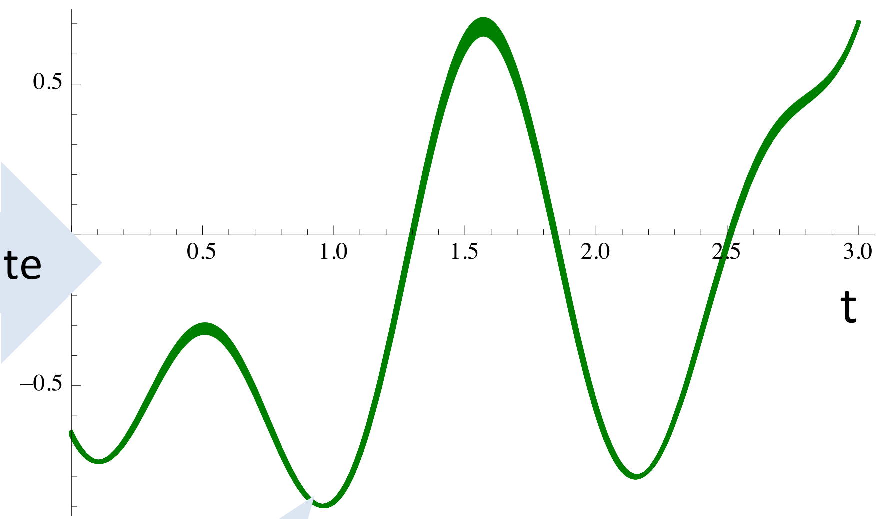
Event time is the positive minimal time satisfying the guard.

Trajectory : $x = -0.5 + 0.2\,t^2 \wedge\ y = -0.3 + \sin(3t) + \varepsilon/100$

Guard: $g(x, y) = x^2 + y^2 - 1 = 0$



Substitute



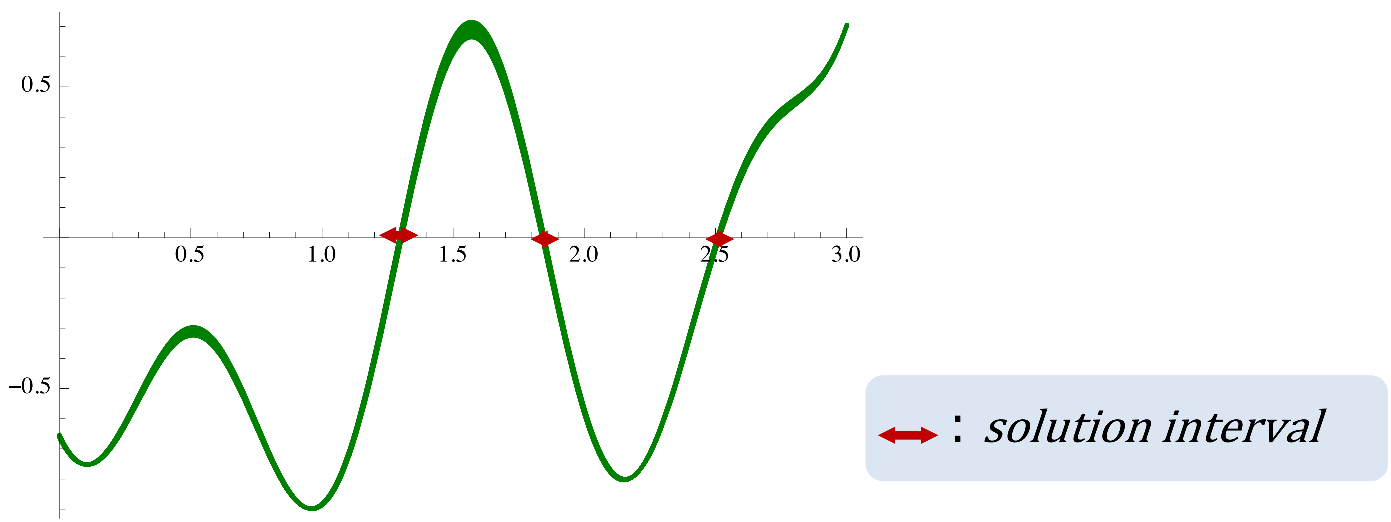
Obtain $f(t, \varepsilon)$

Step 2. Interval Newton Method

Extended version of Newton method

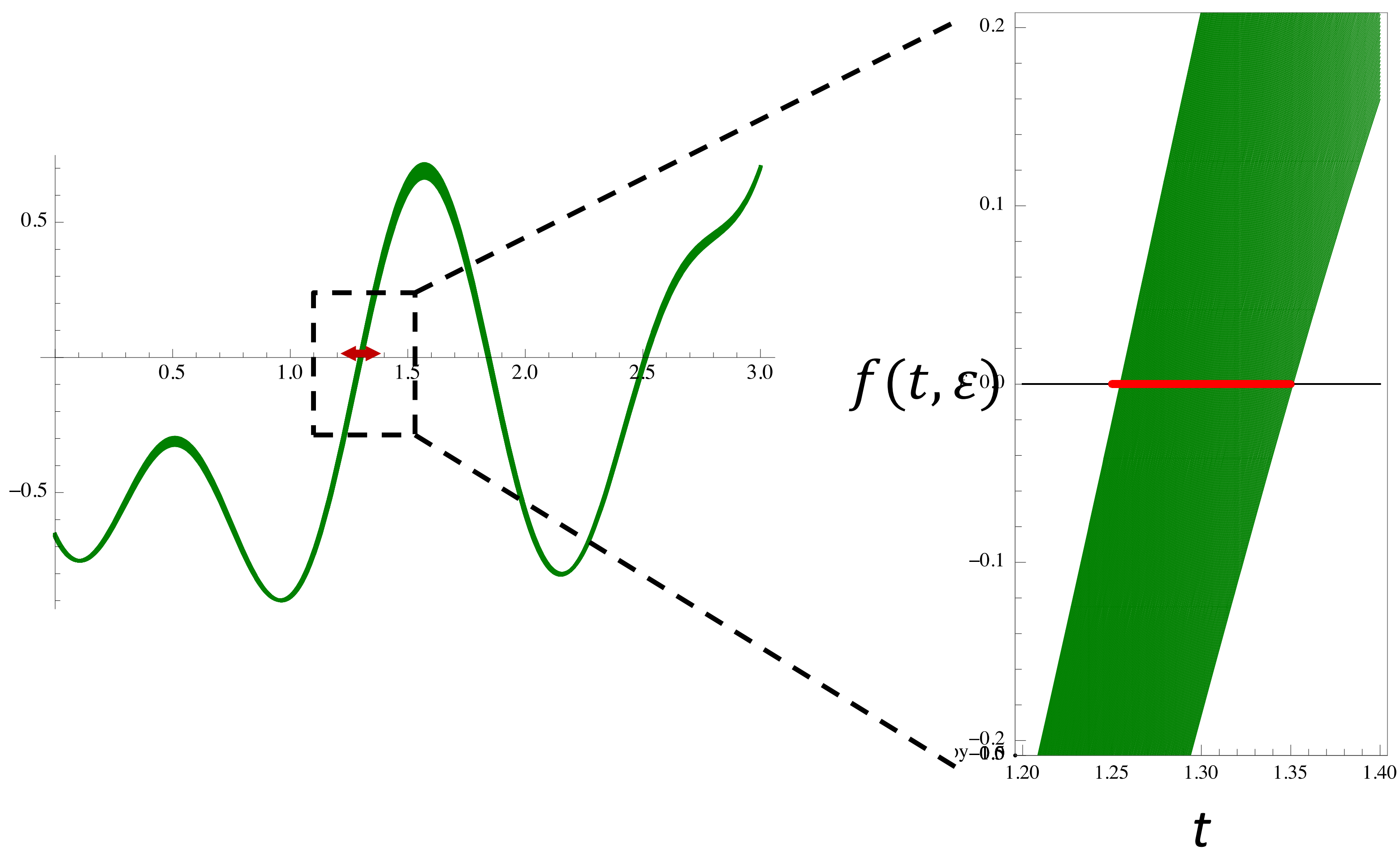
Features:

- Computes over-approximated zero-crossing of $f(t, \varepsilon)$
- Converges quadratically
- Guarantees existence and uniqueness of solution



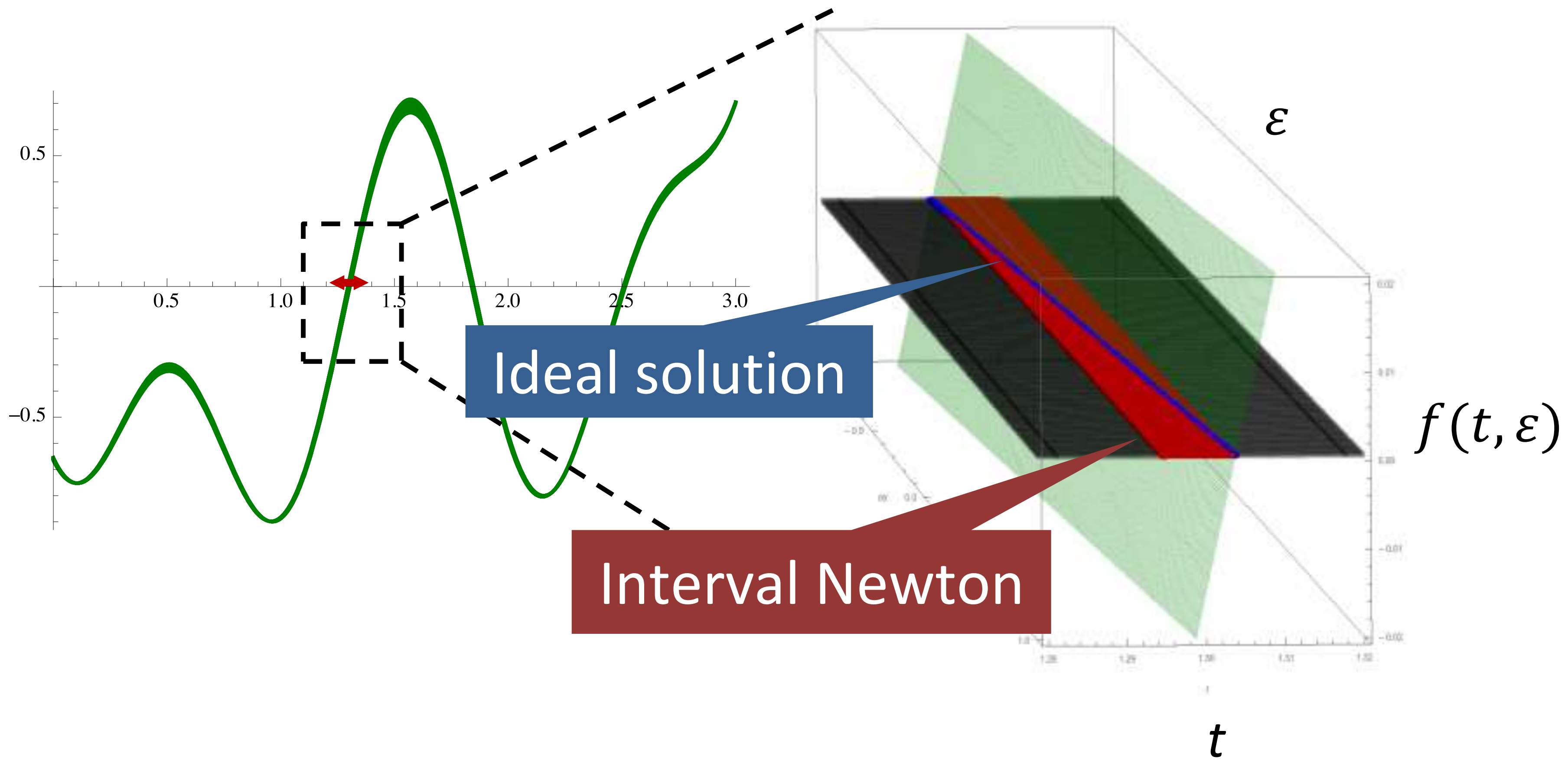
[3] Moore, R. E., Kearfott, R. B., Cloud. M. J.: Society for Industrial and Applied Mathematics, 2009.

Step 2. Solution of Interval Newton Method



- Narrow enough along the time axis

Step 2. Solution of Interval Newton Method



- Narrow enough along the time axis, but
- Not optimal along the parameter axis

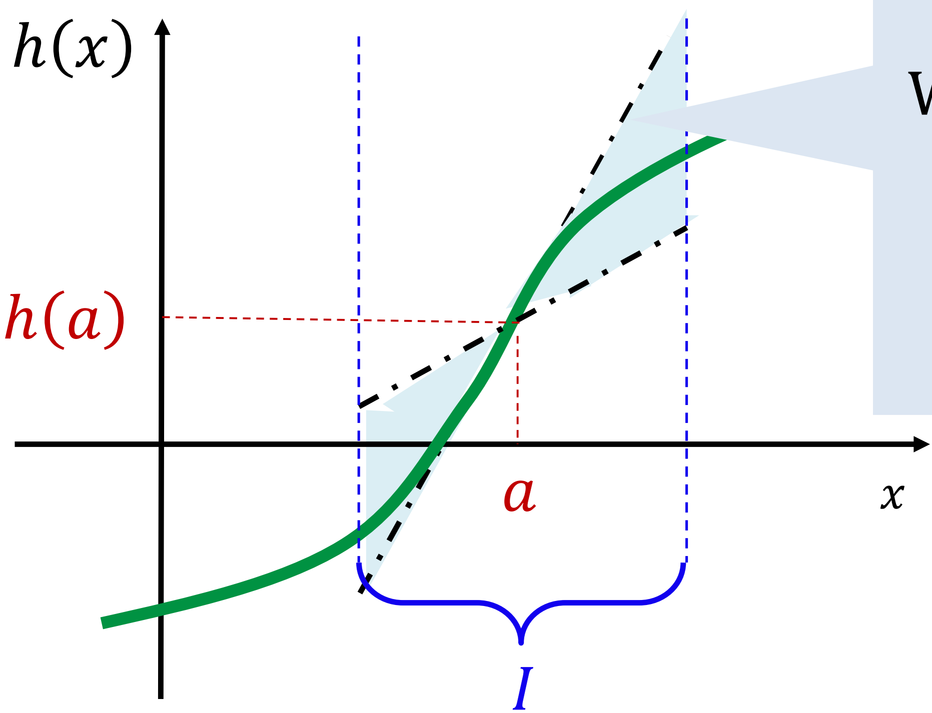
Step 3. Refinement by Mean Value Theorem

Derive **parametrized** solution from solution **interval**

- Compute parametrized over-approximation of $f(t, \varepsilon)$

By mean value theorem for multivariate function

$$[b, a] \subseteq I \Rightarrow h(b) \in h(a) + \nabla h(I) \cdot (b - a)$$



Within I, $h(x)$ is surrounded by the steepest slope and the most moderate slope that passes $(a, h(a))$

Step 3. Refinement by Mean Value Theorem

From $h(b) \in h(a) + \nabla h(I) \cdot (b - a)$,

by replacing $h(x)$ with $f(t, \varepsilon)$, we obtain

T_m is midpoint of T

$$f(t,\varepsilon) \in f(T_m, \varepsilon_m) + \frac{\partial f(T,[-1,1])}{\partial t}(t - T_m) + \frac{\partial f(T,[-1,1])}{\partial \varepsilon}(\varepsilon - \varepsilon_m)$$

$\varepsilon_m = 0$ is midpoint of ε

$$= f(T_m, 0) + \frac{\partial f(T,[-1,1])}{\partial t}(t - T_m) + \frac{\partial f(T,[-1,1])}{\partial \varepsilon}\varepsilon$$

$$=: F(t, \varepsilon)$$

Evaluated to intervals

remaining symbols

Step 3. Refinement by Mean Value Theorem

From $h(b) \in h(a) + \nabla h(I) \cdot (b - a)$,

by replacing $h(x)$ with $f(t, \varepsilon)$, we obtain

T_m is midpoint of T

$$f(t,\varepsilon) \in f(T_m, \varepsilon_m) + \frac{\partial f(T,[-1,1])}{\partial t}(t - T_m) + \frac{\partial f(T,[-1,1])}{\partial \varepsilon}(\varepsilon - \varepsilon_m)$$

ε_m = 0 is midpoint of ε

$$= f(T_m, 0) + \frac{\partial f(T,[-1,1])}{\partial t}(t - T_m) + \frac{\partial f(T,[-1,1])}{\partial \varepsilon}\varepsilon$$

$=: F(t,\varepsilon)$

Evaluated to intervals

Zero-crossing of $F(t,\varepsilon)$ is computed analytically

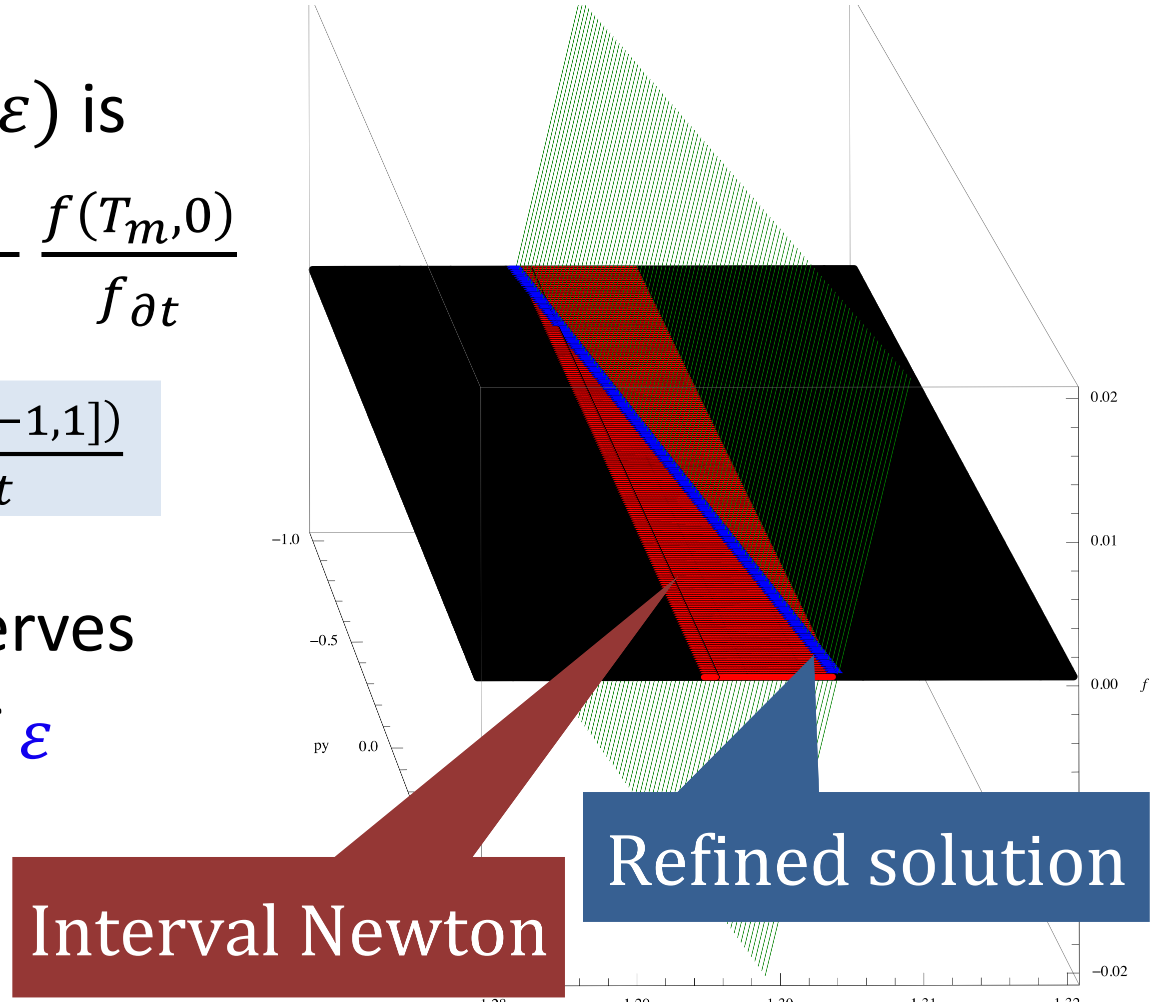
Step 4. Compute parametrized zero-crossing

Zero crossing of $F(t, \varepsilon)$ is

$$t = -\frac{f_{\partial\varepsilon}}{f_{\partial t}} \cdot \boldsymbol{\varepsilon} + T_m - \frac{f(T_m,0)}{f_{\partial t}}$$

abbreviation of $\frac{\partial f(T,[-1,1])}{\partial t}$

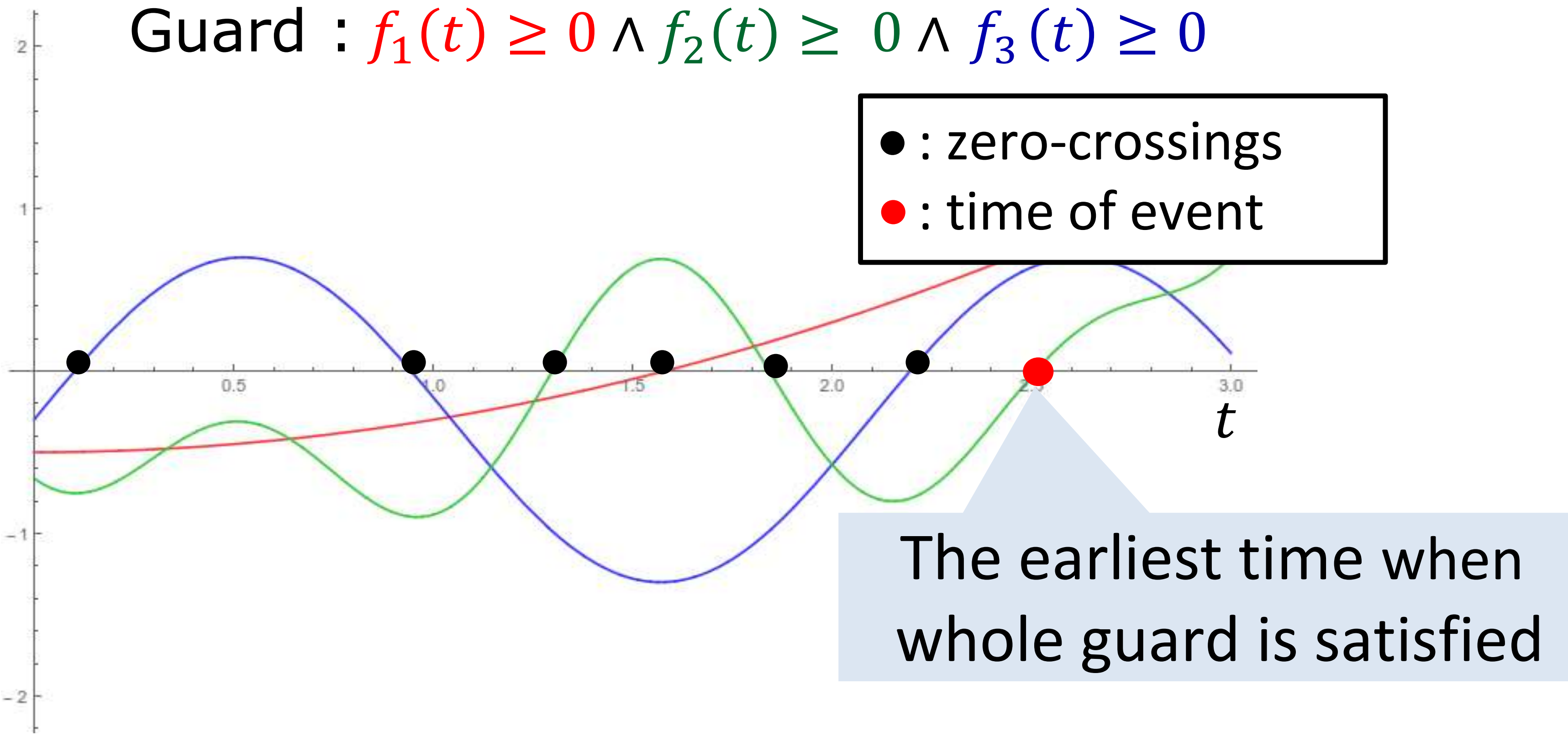
- The solution preserves the linear term of ε



For System of Inequalities

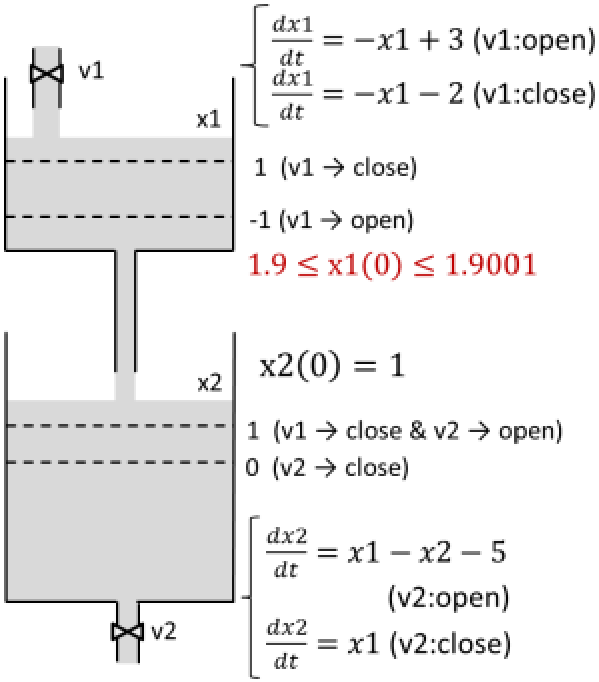
If guards are described by inequalities,
we compute zero-crossings of each atomic condition

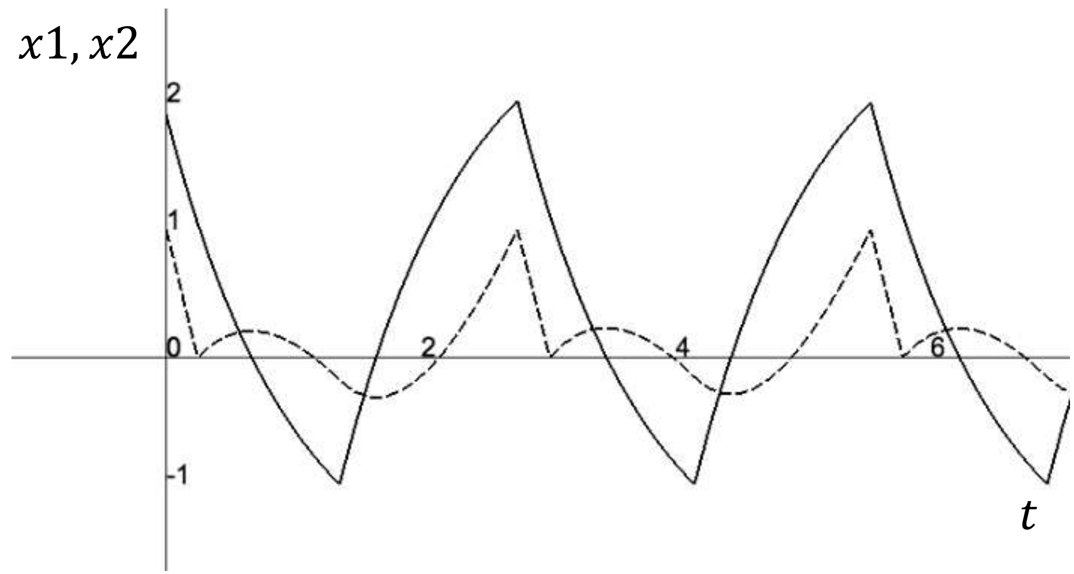
Guard : $f_1(t) \geq 0 \wedge f_2(t) \geq 0 \wedge f_3(t) \geq 0$



Evaluation by Examples

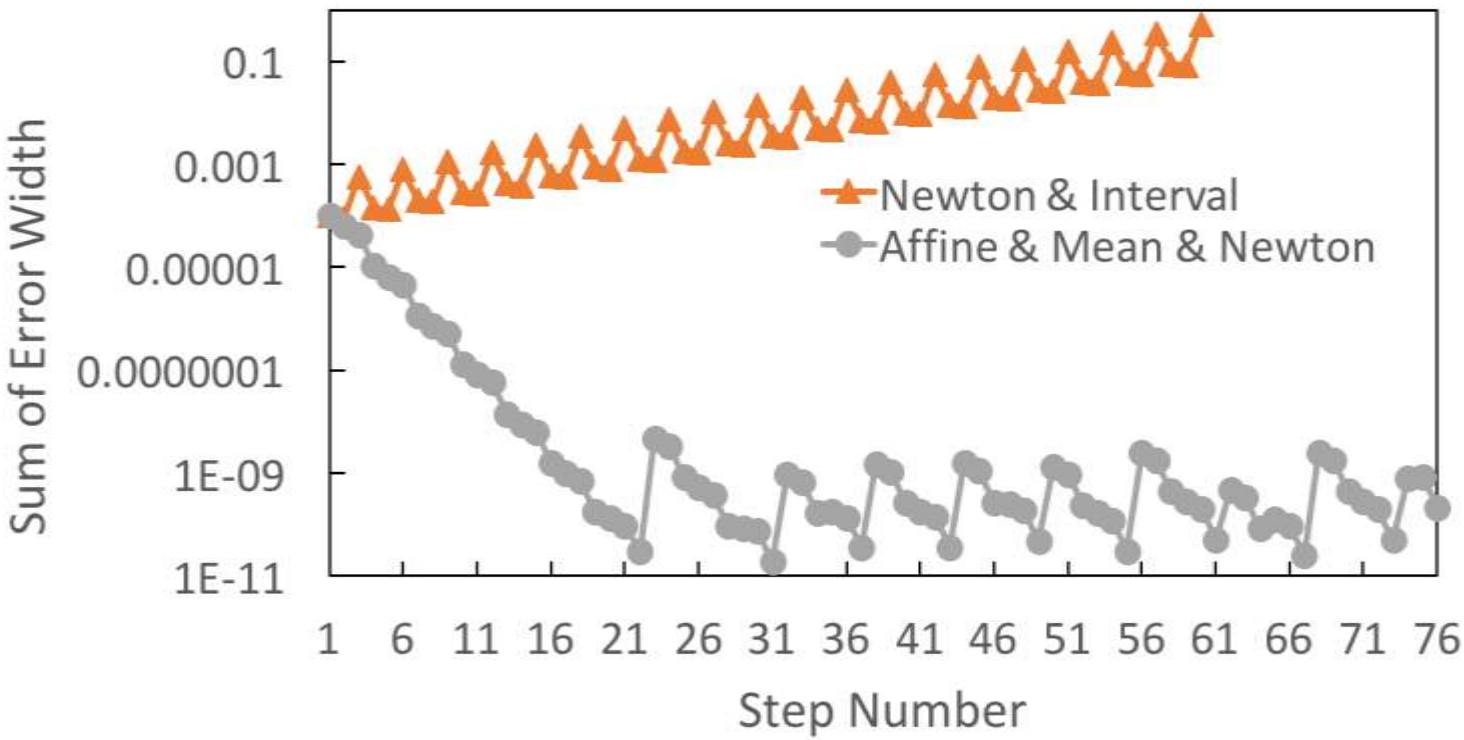
Water Level Control





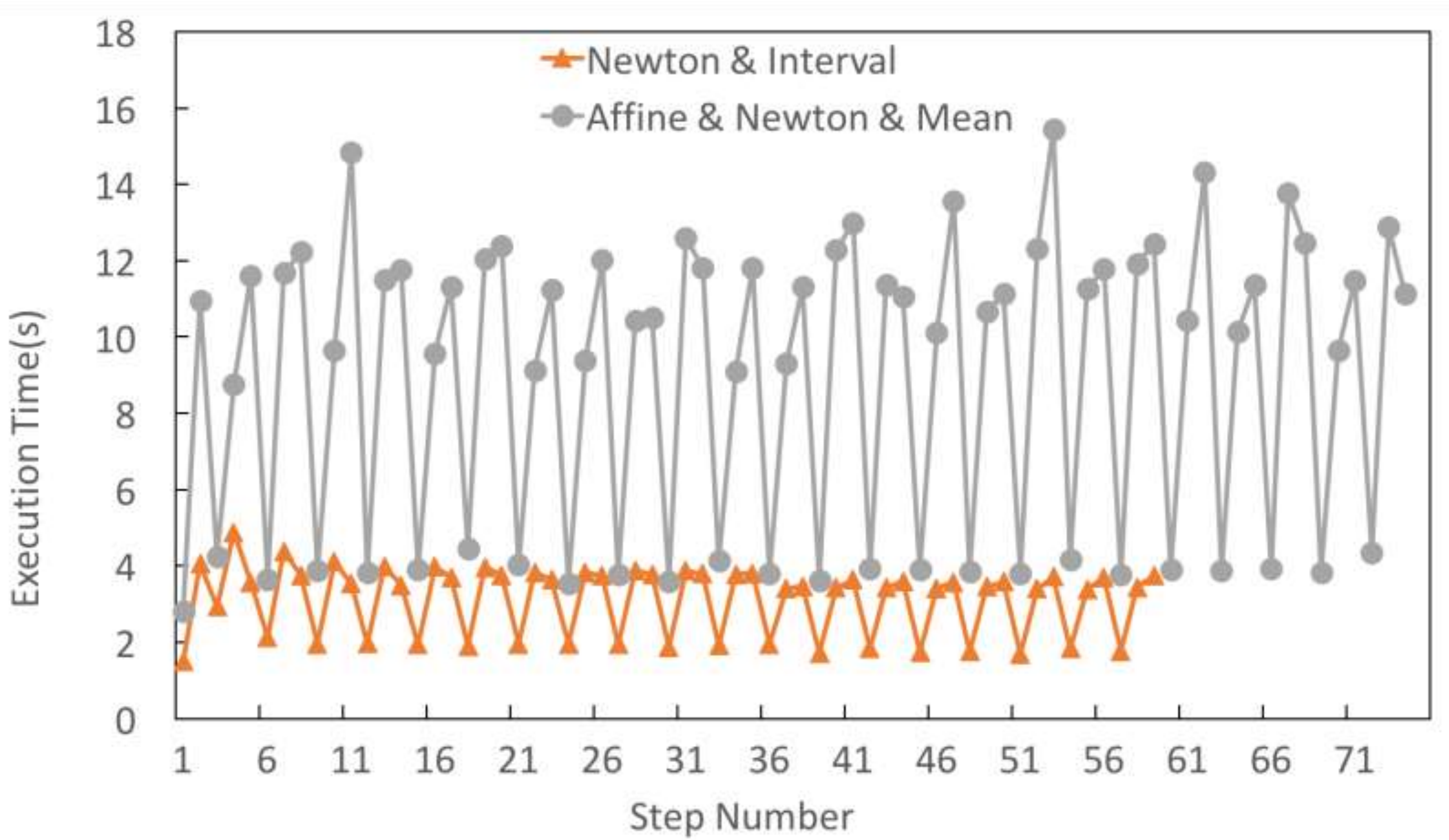
- Compared with naive interval arithmetic
- Preserve 6 symbolic parameters (4 for water level + derivatives, time, additional)

Error width of Water Level Control



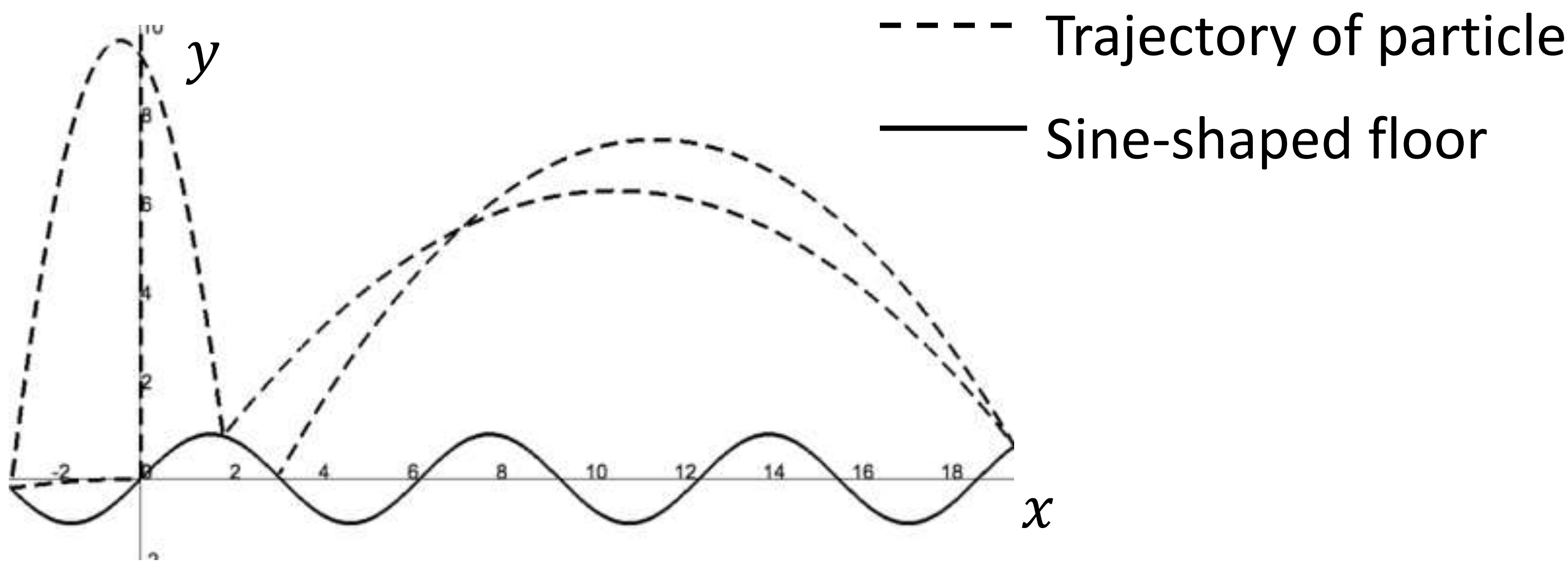
- Error width converged in the proposed method

Execution time of Water Level Control



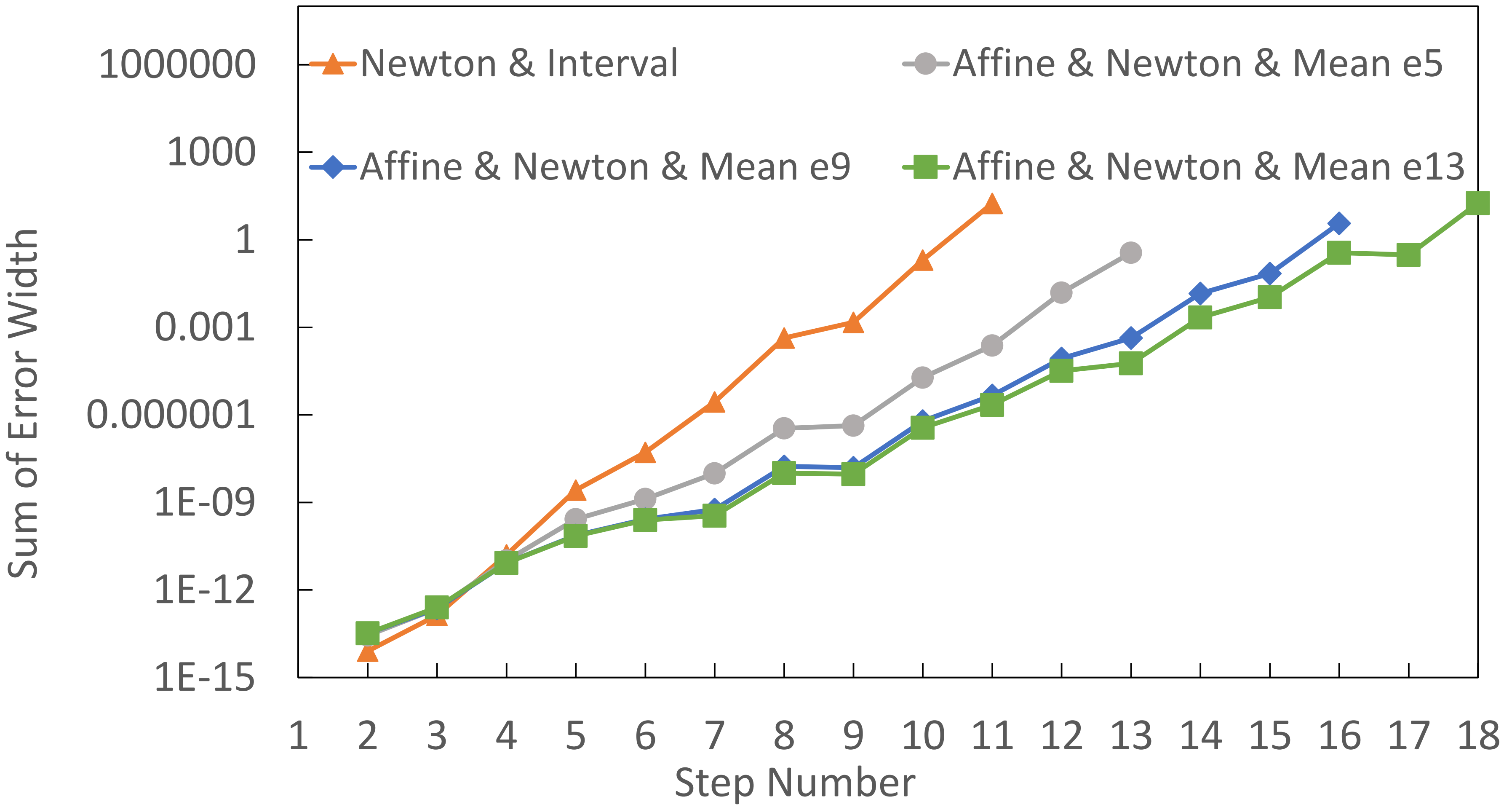
- Execution time is longer than naive interval arithmetic, but did not explode

Bouncing Ball on Sine Wave



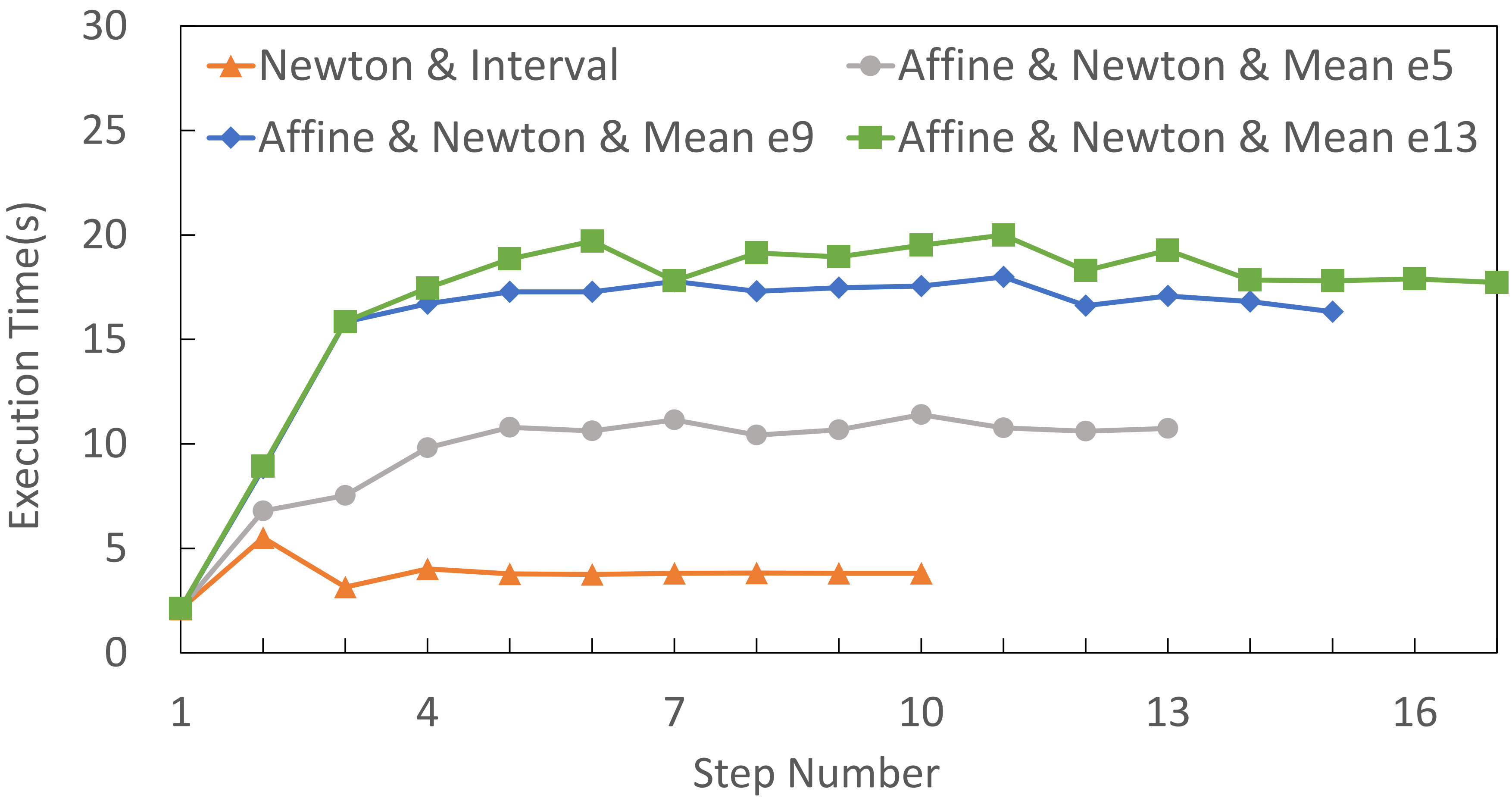
- Compared with naive interval arithmetic
- Preserved {5, 9, 13} parameters

Error width of Bouncing Ball



- Compared with naive interval arithmetic

Execution time of Bouncing Ball



- Tradeoff between error width and execution time

Time for Q&A

Thanks for the attention!